CoGAPS 3.2 Application Note - Supplemental Material

Thomas Sherman, Tiger Gao, and Elana Fertig

July 5, 2019

1 The Standard CoGAPS Algorithm

Before we describe the modifications made to CoGAPS for version 3.2, it is useful to have a full description of the original algorithm to give context for the new work. The input for CoGAPS is a data matrix, $D \in \mathbb{R}^{N \times M}$, an uncertainty matrix, $\sigma \in \mathbb{R}_+^{N \times M}$, and a number of patterns (latent spaces) to learn, K. It factors D into two lower dimensional matrices, $A \in \mathbb{R}_+^{N \times K}$ and $P \in \mathbb{R}_+^{M \times K}$, to fit the model $D \sim \mathcal{N}(AP^T, \sigma)$. Similar to most Bayesian NMF algorithms, CoGAPS uses a Gibbs sampling procedure to iteratively update the values of the A and P matrix. CoGAPS differs from other algorithms through an additional constraint placed on the values of the A and P matrices. Rather than working with values directly stored in the matrix elements, CoGAPS uses a continuous domain of point masses called atoms. These atoms each have a position and a mass. The atomic domain is separated into $N \cdot K$ bins for the A matrix and $M \cdot K$ bins for the P matrix. In this way, each position in the atomic domain maps to a single element of the corresponding matrix. The value of a particular matrix element is equal to the sum of all atom masses in the corresponding bin of the atomic domain. The update steps work directly on the position and mass of the atoms rather than the values of the matrix elements. This allows for a natural way to enforce sparsity in the A and P matrices by controlling the number and average mass of the atoms. While this provides benefits from an analytical point of view, it does put additional burden on both the mathematics and implementation of the CoGAPS algorithm.

1.1 Algorithm Overview

Let $\mathcal{A}$ and $\mathcal{P}$ be the atomic domains for the A and P matrices. Each of the atomic domains is a set of 2-tuples (l, x) where $l \in \mathbb{N}$ is the position of the atom and $x \in \mathbb{R}_+$ is the mass of the atom. In practice we restrict the length of the atomic domain so that $0 \leq l \leq 2^{64}$. Let $|\mathcal{A}|$ and $|\mathcal{P}|$ be the number of atoms in each domain. We also define $\mathcal{A}_{length}$ and $\mathcal{P}_{length}$ to be the size, or number of available positions, of each domain. The size is equal to the closest number to 2^{64} that the number of bins, or matrix elements, evenly divides. This allows each matrix element to have an equal sized portion of the atomic domain mapping to it.

When updating the A and P matrix, CoGAPS works directly with the atoms in $\mathcal{A}$ and $\mathcal{P}$. There are four types of updates that can happen in the atomic domain: 1) Birth - creates a new atom 2) Death - destroys an existing atom 3) Move - changes the location of an existing atom 4) Exchange - swaps the masses of two existing atoms. Each update step starts by randomly selecting one of these 4 changes. In any given update step, either one or two matrix elements will change. In a typical Bayesian NMF algorithm, Gibbs sampling is done by sampling from the conditional posterior distribution of the matrix element itself, i.e. from $p(A_{i,j}|A_{\setminus(i,j)}, P, D, \sigma)$ or $p(P_{i,j}|P_{\setminus(i,j)}, A, D, \sigma)$. CoGAPS works in a similar manner, except in this case we aren't sampling a new value for the matrix element, but rather some value to change it by, i.e. we sample from $p(\Delta A_{i,j}|A, P, D, \sigma)$ or $p(\Delta P_{i,j}|P, A, D, \sigma)$. This allows us to do Gibbs sampling directly on the atoms in $\mathcal{A}$ and $\mathcal{P}$.

The sparsity of the atomic domain is determined by the relative probability of a Birth or a Death step. This is controlled by a user parameter, α. This parameter determines the equilibrium point where Birth and Death steps will be equally likely, and so it controls the expected number of atoms in each domain. The parameter, λ, influences the expected mass of each atom and is a function of α:

$$\lambda = \alpha \sqrt{\frac{K}{\overline{D}}} \tag{1}$$

Where K is the number of patterns input by the user and $\overline{D}$ is the average value of the data matrix. The prior distribution put on the mass of an atom is an exponential with parameter λ.

Update steps are run in batches, alternating between the A and P matrix. An iteration of CoGAPS is defined as a batch of updates for each the A and P matrix. The algorithm runs in two phases, calibration (burn-in) and sampling. Each of these phases is run for a pre-specified number of iterations, input by the user. The goal is to pick a number of iterations large enough for the calibration step of the algorithm to converge. If convergence is hit, then in the sampling phase, CoGAPS will truly be taking samples from the posterior distribution, $p(A, P|D, \sigma)$. These samples are averaged to get the final values for A and P. This is the Bayes minimum mean square error estimator.

1.2 Important Calculations

Before we describe the specifics of the update steps, it's good to note some of the important calculations that are made in each update. We define these calculations here since they are critical to the performance of the algorithm and so much of the optimization work is centered around computing these values. They are particularly important for the sparse data optimization presented in a later section. A full derivation is presented in the Appendix.

1.2.1 Single Matrix Element Calculations

These calculations are relevant when a single matrix element is being changed (such as in birth or death). Let r, q be the row and column of the changed element and let $^{(A)}$ or $^{(P)}$ denote which matrix is being changed.

$$s^{(A)}_{r,q} = \sum_j \frac{P^2_{jq}}{2\sigma^2_{rj}} \qquad s^{(P)}_{r,q} = \sum_i \frac{A^2_{iq}}{2\sigma^2_{ir}}$$

$$s\mu^{(A)}_{r,q} = \sum_j \frac{P_{jq}(D_{rj} - (AP^T)_{rj})}{2\sigma^2_{rj}} \qquad s\mu^{(P)}_{r,q} = \sum_i \frac{A_{iq}(D_{ir} - (AP^T)_{ir})}{2\sigma^2_{ir}}$$

In this case, we want to make the same $s\mu$ calculation, but with the assumption that a change x has already been made to the (r, q) matrix element.

$$s\mu^{(A)}_{r,q,x} = \sum_j \frac{P_{jq}(D_{rj} - (AP^T)_{rj} - xP_{jq})}{2\sigma^2_{rj}} \qquad s\mu^{(P)}_{r,q,x} = \sum_i \frac{A_{iq}(D_{ir} - (AP^T)_{ir} - xA_{iq})}{2\sigma^2_{ir}}$$

1.2.2 Multiple Matrix Element Calculations

These calculations are relevant when two matrix elements are being changed at the same time (such as in move or exchange). Let r_1, q_1 be the row and column of the first element and r_2, q_2 be the second element.

$$s\mu^{(A)}_{r_1,r_2,q_1,q_2} = s\mu^{(A)}_{r_1,q_1} - s\mu^{(A)}_{r_2,q_2} \qquad s\mu^{(P)}_{r_1,r_2,q_1,q_2} = s\mu^{(P)}_{r_1,q_1} - s\mu^{(P)}_{r_2,q_2}$$

If $r_1 = r_2$:

$$s^{(A)}_{r_1,r_2,q_1,q_2} = \sum_j \frac{(P_{jq_1} - P_{jq_2})^2}{2\sigma^2_{r_1 j}} \qquad s^{(P)}_{r_1,r_2,q_1,q_2} = \sum_i \frac{(A_{iq_1} - A_{iq_2})^2}{2\sigma^2_{ir_1}}$$

If $r_1 \neq r_2$:

$$s^{(A)}_{r_1,r_2,q_1,q_2} = s^{(A)}_{r_1,q_1} + s^{(A)}_{r_2,q_2} \qquad s^{(P)}_{r_1,r_2,q_1,q_2} = s^{(P)}_{r_1,q_1} + s^{(P)}_{r_2,q_2}$$

Algorithm 1: CoGAPS

Input: $D, \sigma \in \mathbb{R}^{N \times M}$, $n, K \in \mathbb{N}$, $\alpha \in \mathbb{R}_+$
Output: $A \in \mathbb{R}_+^{N \times K}$, $P \in \mathbb{R}_+^{M \times K}$

`/* Initialize the A and P matrices along with their atomic domains */`
$A := 0_{N,K}, P := 0_{M,K}$
$\mathcal{A} := \varnothing, \mathcal{P} := \varnothing$

`/* Run the calibration phase of the algorithm */`
for $i = 1...n$ **do**
  temp $= \min(1, 2i/n)$
  $n_A \leftarrow Poisson(|\mathcal{A}|), n_P \leftarrow Poisson(|\mathcal{P}|)$
  do update$(A, \mathcal{A}$, temp$)$ n_A **times**
  do update$(P, \mathcal{P}$, temp$)$ n_P **times**
end

`/* Run the sampling phase of the algorithm */`
$A_{final}, P_{final} := 0$
for $i = 1...n$ **do**
  $n_A \leftarrow Poisson(|\mathcal{A}|), n_P \leftarrow Poisson(|\mathcal{P}|)$
  do update$(A, \mathcal{A}, 1)$ n_A **times**
  do update$(P, \mathcal{P}, 1)$ n_P **times**
  A_{final} += A, P_{final} += P
end

`/* Return the estimate of the mean of the posterior distribution */`
return A_{final}/n, P_{final}/n

Algorithm 2: Update

`/* M is either A or P, M is the assoicated atomic domain */`
Input: $M, \mathcal{M}$, temp

$u_1 \leftarrow Uniform(0,1), u_2 \leftarrow Uniform(0,1)$
if $u_1 < 1/2$ **then**
  $p := \dfrac{|\mathcal{M}|\mathcal{M}_{length}}{|\mathcal{M}|\mathcal{M}_{length} + \alpha|M|(\mathcal{M}_{length} - |\mathcal{M}|)}$
  if $u_2 < p$ **then**
    death$(M, \mathcal{M}$, temp$)$
  else
    birth$(M, \mathcal{M}$, temp$)$
  end
else
  if $u_2 < 1/2$ **then**
    move$(M, \mathcal{M}$, temp$)$
  else
    exchange$(M, \mathcal{M}$, temp$)$
  end
end

1.3 The Update Step

The update step enforces the mathematics underlying CoGAPS and so it forms the core, most important component of the algorithm. There are two important algorithms contained in this step, one is responsible for choosing which elements of A or P to change, and one is responsible for choosing the magnitude of the change. The first is important since there are additional constraints in the atomic domain so we can't arbitrarily choose an order in which to update the elements and the second allows us to enforce the mathematics of gibbs sampling so that we converge to a solution.

There are four types of updates: birth, death, move, and exchange. The probability of an update step being any of these is described in **Algorithm 1**. In this section, we will describe how each step works. We break down each step into two parts: the proposal and the evaluation. This distinction becomes important for when we present the asynchronous updating scheme in the next section. The proposal phase determines which atoms are affected and the evaluation phase determines how large the effect is, and whether or not to accept certain changes.

Note that we use M and $\mathcal{M}$ here for the matrix and atomic domain, since these steps are the same for either the A or the P matrix.

1.3.1 Birth

In this type of update we are creating a new atom in a random location of the atomic domain.

Proposal

The proposal part of this step involves selecting a random, open location for the atom.

$l \leftarrow \mathcal{U}\{0, \mathcal{M}_{length}\}$ such that $(l, \cdot) \notin \mathcal{M}$

Let $l \mapsto r, q$, where r, q are the corresponding row and column in the matrix.

Evaluation

We want to sample a mass for this atom x. This corresponds to a change in the $M_{r,q}$ element, i.e. $\Delta M_{r,q} = x$. Thus, we want to sample x from $p(\Delta M_{r,q}|A, P, D, \sigma)$. In the Appendix, we provide the derivation to show that this distribution is a truncated normal with the following parameters:

$$\mathcal{TN}_{(0,\cdot)}\left(\frac{2s\mu_{r,q}^{(M)} - \lambda}{2s_{r,q}^{(M)}}, \frac{1}{2\sqrt{s_{r,q}^{(M)}}}\right)$$

We sample x from this distribution and add the new atom, (l, x), to $\mathcal{M}$.

1.3.2 Death

In this type of update, we select an atom at random and remove it from the atomic domain. In practice, we take an additional step that helps the algorithm converge faster. Rather than just removing the atom outright, we give it a chance to re-sample a new mass and then keep that atom with some probability based on how the new mass effects the likelihood, $p(D|A, P, \sigma)$.

Proposal

In order to propose a death step, we only need to select an atom uniformly at random. Let (l, x_0) be the location and mass of this atom, and let $l \mapsto r, q$ where r, q are the corresponding row and column of the matrix.

Evaluation

When evaluating the death step, we need to first remove the atom from the atomic domain and then sample

a new mass in the same location. Rather than actually removing the mass and updating the matrix, we can instead directly compute $p(\Delta M_{r,q}|x_0, A, P, D, \sigma)$ where x_0 is the original mass that has been removed. In the Appendix, we provide the derivation to show that this distribution is a truncated normal with the following parameters:

$$\mathcal{TN}_{(0,\cdot)}\left(\frac{2s\mu^{(M)}_{r,q,x_0} - \lambda}{2s^{(M)}_{r,q}}, \frac{1}{2\sqrt{s^{(M)}_{r,q}}}\right)$$

We sample x from this distribution to get a new atom, (l, x). However, we only add it to $\mathcal{M}$ with some probability. Otherwise, the atom is removed from the atomic domain completely. The probability of keeping this new atom is $\exp\{\Delta\mathcal{L}(x, x_0)\}$, where $\Delta\mathcal{L}(x, x_0)$ is defined as the log-likelihood ratio:

$$\Delta\mathcal{L}(x, x_0) = \log(p(D|\Delta M_{r,q} = x - x_0, \sigma, A, P)) - \log(p(D|\Delta M_{r,q} = -x_0, \sigma, A, P)) = \frac{x(s\mu^{(M)}_{r,q,x_0} - xs^{(M)}_{r,q})}{2}$$

A full derivation is presented in the Appendix.

1.3.3 Move

In this type of update, we are moving an atom from one location to another. We restrict the move so that the atomic domain is not re-ordered, i.e. an atom can only move between it's left and right neighbors.

Proposal

We select an atom uniformly at random. Let (l_0, x) be the location and mass of this atom. Let l_{left} and l_{right} be the locations of the left and right neighbors of this atom. If the neighbor does not exist then $l_{left} = 0$ and $l_{right} = \mathcal{M}_{length}$. We then sample a new location l from the uniform distribution:

$l \sim \mathcal{U}\{l_{left}, l_{right}\}$

Let $l_0 \mapsto r_0, q_0$ and $l \mapsto r, q$ be the corresponding row and column each location maps to.

Evaluation

If $r_0 = r$ and $q_0 = q$:

Automatically accept the change as there is no effect on the matrix, simply update the atom's position.

If $r_0 \neq r$ or $q_0 \neq q$:

Accept the move with probability, $\exp\{\Delta\mathcal{L}(l_0, l)\}$, where $\Delta\mathcal{L}(l_0, l)$ is defined as the log-likelihood ratio:

$$\Delta\mathcal{L}(l, l_0) = \log(p(D|\Delta M_{r,q} = x, \Delta M_{r_0,q_0} = -x, \sigma, AP)) - \log(p(D|\sigma, A, P)) = \frac{x(s\mu^{(M)}_{r_0,r,q_0,q} - xs^{(M)}_{r_0,r,q_0,q})}{2}$$

1.3.4 Exchange

In this type of update, we are exchanging mass between two atoms. First an atom is selected at random. Then we use a Gibbs sampling approach to determine how much mass to exchange between this atom and it's right neighbor.

Proposal

We select an atom uniformly at random. Let (l_1, x_1) be this atom and let (l_2, x_2) be its right neighbor. If the right neighbor does not exist (i.e. l_1 is the right-most atom) then set the right neighbor equal to the left-most atom. Let $l_1 \mapsto r_1, q_1$ and $l_2 \mapsto r_2, q_2$ be the corresponding row and column each location maps to.

Evaluation

If $r_1 \neq r_2$ or $q_1 \neq q_2$:

We want to sample the mass increase to the first atom, let this be x (i.e. $\Delta x_1 = x$). Since we are exchanging mass, we must preserve the total mass between both atoms. To accomplish this we must have $x \in (-x_1, x_2)$. At the end of this step we will set the new masses to be $x_1 \to x_1 + x$ and $x_2 \to x_2 - x$. We want to sample x from $p(\Delta M_{r_1,q_1} = x, \Delta M_{r_2,q_2} = -x | D, \sigma, A, P)$. In the Appendix, we provide the derivation to show that this distribution is a truncated normal with the following parameters:

$$\mathcal{TN}_{(-x_1,x_2)}\left(\frac{s\mu^{(M)}_{r_1,r_2,q_1,q_2}}{s^{(M)}_{r_1,r_2,q_1,q_2}}, \frac{1}{2\sqrt{s^{(M)}_{r_1,r_2,q_1,q_2}}}\right)$$

We sample x from this distribution and set $x_1 \to x_1 + x$ and $x_2 \to x_2 - x$.

If $r_1 = r_2$ and $q_1 = q_2$:

In this case, there is no change to the matrix elements or the structure of the atomic domain so it is safe to ignore this update.

1.4 Annealing Temperature

When the algorithm is calibrating, we use a strategy known as simulated annealing. This allows the initial samples to be dominated by the prior distributions rather than the likelihood. Let τ be the temperature, then in all cases we have $s \leftarrow \tau s$ and $s\mu \leftarrow \tau s\mu$. Eventually we reach $\tau = 1$, in which case s and $s\mu$ are calculated exactly as described above.

2 Asynchronous Update Steps

The previous section outlined the standard CoGAPS algorithm and showed how the update steps of the algorithm work in detail. It is important to note that, in the standard algorithm, these update steps must be run sequentially. The proposal phase of each update step requires knowledge about the current state of the atomic domain. If the updates were run in parallel, it wouldn't be possible to know the current state of the domain and the algorithm would be fundamentally different. The evaluation phase, however, is not so restricted. This phase is largely just a large matrix calculation. If two update steps involve unrelated segments of the matrices, then there's no reason the evaluation phase can't be run in parallel. Conveniently, the proposal phase is relatively inexpensive to compute when compared to the evaluation phase. This concept is the foundation of the asynchronous version of CoGAPS. The proposal phase of the update steps are run sequentially in order to build up a queue of "independent" proposals which are then evaluated in parallel.

The details of the asynchronous algorithm come down to defining independent proposals. We want two proposals to be independent only if they can be evaluated in any order without changing the state of the algorithm (i.e. the atomic domain and matrix values). If we are able to do that, then the algorithm is straightforward: proposal as many updates as possible until there is some dependency, then evaluate all the independent proposals in parallel. If our definition is correct, this will result in the same state of the algorithm as if we processed everything sequentially. Since it is difficult to define what makes proposals independent, we will instead make an exhaustive list of the conflicts that could prevent us from adding a new proposal to the queue. The next three sections explore these conflicts.

2.1 Conflict in Matrix Calculations

The first type of conflict arises in the calculations of s and $s\mu$. Both of the quantities depend on the values in the A and P matrices. For the sake of example, say that we are proposing a change to $A_{r,q}$. In this case, we will calculate $s^{(A)}_{r,q}$ and $s\mu^{(A)}_{r,q}$ which both depend of the r^{th} row of AP^T. If we accept a change to $A_{r,q}$ we will also be changing the entire r^{th} row of AP^T and this is where the conflict happens. If there are two proposals in the same row of A or P, then the outcome of one will effect the calculations of s and $s\mu$ for

the other. Therefore, any two proposals that effect the same rows are not independent. To check for this, we keep a table of all the rows that are currently used by proposals in the queue. If any new proposal has an overlap, we stop the queue and move on to the evaluation phase. Surprisingly, this conflict accounts for nearly all the possible conflicts that could arise, however there are some corner cases described in the next section.

2.2 Atomic Domain Ordering and Finding Neighbors

Checking for a matrix conflict will take care of most of the conflicts that can occur between proposals, however there is one type of update that can be problematic without causing any matrix conflicts. When doing a move step, we select an atom at random and move it somewhere between it's neighbors. That means we must be able to calculate the positions of both neighbors. Most of the time, neighboring atoms will map to matrix elements in the same row, but this is not guaranteed to be the case - especially since the atomic domain can be very sparse. Therefore, it is possible that the starting location and proposed new location for a moved atom could map to matrix elements in a different row than the neighboring atoms. In this case, it would be possible for the neighbors to be involved in a previously queued update step but not have any conflicts in the matrix calculations. To account for this, during a move step we must explicitly check that the neighboring atoms are not involved in any previously queued updates.

There is additional type of conflict that can arise directly in the atomic domain during the birth step. Any queued death step can open up more locations for future atoms to be created in. The probability of a location chosen at random being already occupied is astronomically small (in practice the atomic domain has 2^{64} possible locations). The chance that not only was this location occupied, but the atom also was removed in the last handful of updates is so small that we ignore this conflict. New atoms are created among all the available locations, not counting ones that could be made available from the current queue.

2.3 Birth/Death Decision

Direct conflicts between update steps aren't the only way that the queue can be stopped. It is possible to run into an issue before we even decide what type of update to queue next. When deciding between a birth and death step, we need to know the current size of the atomic domain. This allows the algorithm to enforce a sparsity constraint by tightly controlling the total number of atoms. Every time a proposed update is queued, it becomes less clear what the total number of atoms will be. At update t if we know there n atoms, and we propose a death step, then at $t+1$ there are between $n-1$ and n atoms. The more updates that are queued up, the larger this range becomes.

Fortunately, the function that determines the probability of death vs birth is a monotonic function of the total number of atoms. Let $P_{death}(n)$ be the death probability when there are n atoms and let n_t be the total number of atoms at time t. If we have evaluated all updates up to time t then we know n_t exactly. If we propose k more updates however, we don't know what n_{t+k} is. Based on the types of the proposed updates we can calculate a range for this value. Let N_{deaths} and N_{births} be the number of deaths and births proposed in the current queue. Then we know that $n_t - N_{deaths} + N_{births} \leq n_{t+k} \leq n_t + N_{births}$. We therefore know that $P_{death}(n_t - N_{deaths} + N_{births}) \leq P_{death}(n_{t+k}) \leq P_{death}(n_t + N_{births})$.

When making the decision between a birth step and a death step at time t, we draw $u \sim \mathcal{U}(0,1)$. If $u < P_{death}(n_t)$ we propose a death step, otherwise a birth step. For making this decision k proposals into the future, we have three possible outcomes instead of two. If $u < P_{death}(n_t - N_{deaths} + N_{births})$ then we know we have a death step. If $u > P_{death}(n_t + N_{births})$ then we know we have a birth step. If u is between these two values then we can't make a decision and the result is indeterminate. We must first evaluate the current queue in order to get the exact value for n_{t+k}. So, in this case the queue stops being populated and we move on to the evaluation phase.

3 Sparse Data Optimization

The previous sections have discussed strategies improving the run time of CoGAPS, but have done nothing to address the amount of memory used by the algorithm. Memory overhead is one of the most significant

problems facing the analysis of large single-cell datasets. Fortunately, these datasets tend to be extremely sparse and so there's an opportunity for reducing the memory overhead of the algorithm by storing the data in sparse data structures. Computing the values of s and $s\mu$ on sparse data structures however is not straightforward. It's no use improving the memory overhead of CoGAPS if we cripple the run time. It is therefore necessary to derive a sparse-friendly version of the s and $s\mu$ calculations.

3.1 Sparse Matrix Calculations

In this case we will derive the equations just for the A matrix, since the derivation for the P matrix is identical except for the labels. We first define the non-zero indices of a given row of the data as γ and make an assumption about the standard deviation, σ:

$$\text{Let } \gamma(i) = \{j : D_{ij} > 0\} \text{ and } \sigma_{ij} = \begin{cases} \sigma_0 D_{ij} & \text{if } D_{ij} > 0 \\ \sigma_0 & \text{if } D_{ij} = 0 \end{cases}$$

In the Appendix we show the full derivation of the following identities:

$$s^{(A)}_{r,q} = \sum_j \frac{P^2_{jq}}{2\sigma^2_{rj}} = \frac{1}{\sigma^2_0}\sum_j P^2_{jq} + \frac{1}{\sigma^2_0}\sum_{j\in\gamma(r)}\left(\frac{P^2_{jq}}{D^2_{rj}} - P^2_{jq}\right)$$

$$s\mu^{(A)}_{r,q} = \sum_j \frac{P_{jq}(D_{rj} - \sum_k A_{rk}P_{jk})}{2\sigma^2_{rj}} = \frac{-\sum_k A_{rk}\sum_l P_{lq}P_{lk}}{\sigma^2_0} + \frac{1}{\sigma^2_0}\sum_{j\in\gamma(r)}\frac{P_{jq}}{D_{rj}} + \left(P_{jq} - \frac{P_{jq}}{D^2_{ij}}\right)\sum_k A_{rk}P_{jk}$$

$$s\mu^{(A)}_{r,q,x} = \sum_j \frac{P_{jq}(D_{rj} - \sum_k A_{rk}P_{jk} - xP_{jq})}{2\sigma^2_{rj}}$$

$$= \frac{-x\sum_j P^2_{jq} - \sum_k A_{rk}\sum_j P_{jq}P_{jk}}{\sigma^2_0} + \frac{1}{\sigma^2_0}\sum_{j\in\gamma(r)}\frac{P_{jq}}{D_{rj}} + \left(P_{jq} - \frac{P_{jq}}{D^2_{ij}}\right)\left(\sum_k A_{rk}P_{jk} + xP_{jq}\right)$$

(When $r = r_1 = r_2$)

$$s^{(A)}_{r_1,r_2,q_1,q_2} = \sum_j \frac{(P_{jq_1} - P_{jq_2})^2}{2\sigma^2_{rj}} = s^{(A)}_{r,q_1} + s^{(A)}_{r,q_2} - \frac{2\sum_j P_{jq_1}P_{jq_2}}{\sigma^2_0} + \frac{2}{\sigma^2_0}\sum_{j\in\gamma(r)}\left(P_{jq_1}P_{jq_2} - \frac{P_{jq_1}P_{jq_2}}{D^2_{rj}}\right)$$

Note that we can longer use the AP^T matrix in the calculations since it is not feasible to cache it when working with large, sparse datasets. Even when D is very sparse, the product of A and P will not be. In fact, AP^T is nearly all non-zero is most real world cases. Storing this large of a matrix defeats the entire purpose of the optimization, so we must compute $A \cdot P$ on the fly.

3.2 Runtime Complexity

Each of these calculations follow a similar pattern: there is some initial term and then a sum over the non-zero indices of the data. Without the optimization, we would have to sum over all the indices of the data. Therefore, as long as the initial term doesn't take too long to compute, the full computation should be faster than the standard computation by a factor of the data sparsity. In this section, we will explore the runtime complexity and computational strategies for the initial terms.

$$s^{(A)}_{r,q} \to \frac{1}{\sigma^2_0}\sum_j P^2_{jq}$$

This term only depends on the P matrix which is constant while A is being updated, so it can be pre-computed ahead of a batch update step for A. To compute this term for all values of q requires $O(MK)$ time where M is the number of samples and K is the number of patterns. A batch of updates can be done to A is $O(NMK)$ time where N is the number of features. Thus creating a lookup table for this initial term has a negligible cost when compared to the full batch of updates. Once we have a lookup table, then computing the initial term takes $O(1)$.

$$s\mu^{(A)}_{r,q} \to \frac{-\sum_k A_{rk} \sum_j P_{jq} P_{jk}}{\sigma_0^2}$$

The $\sum_j P_{jq} P_{jk}$ term only depends on P so we can also make a lookup table for this value. Creating this lookup table takes $O(MK^2)$ time, whereas the full batch of updates takes $O(MNK)$. In practice, $K << N$ so this is an acceptable overhead to incur. Once we have a lookup table, the initial term takes $O(K)$ to compute, which makes the update step go from $O(MK)$ to $O(MK+K)$ which again is acceptable since $K << M$.

$$s\mu^{(A)}_{r,q,x} \to \frac{-x \sum_j P^2_{jq} - \sum_k A_{rk} \sum_j P_{jq} P_{jk}}{\sigma_0^2}$$

We've already created a lookup table for both $\sum_j P^2_{jq}$ and $\sum_j P_{jq} P_{jk}$ so there is no additional cost for this step, and the remaining computation takes the same amount of time as the previous quantity, $s\mu^{(A)}_{r,q}$ so no additional work needs to be done to show this incurs an acceptable overhead.

$$s^{(A)}_{r_1,r_2,q_1,q_2} \to s^{(A)}_{r,q_1} + s^{(A)}_{r,q_2} - \frac{2 \sum_j P_{jq_1} P_{jq_2}}{\sigma_0^2}$$

While it may seem that $s^{(A)}_{r,q_1} + s^{(A)}_{r,q_2}$ are part of the initial term, both of these quantities can be computed alongside the main term, so no additional run time cost is incurred. The $\sum_j P_{jq_1} P_{jq_2}$ term has already been addressed above, so there is no additional work needed.

4 Appendix

In this section we provide a full derivation of all equations.

4.1 Likelihood of a single change

The likelihood of a particular change to the A matrix, ΔA_{kl}, is given by:

$$p(D|\Delta A_{kl}, \sigma, AP) = \prod_i \prod_j p(D_{ij}|\Delta A_{kl}, \sigma_{ij}, (AP)_{ij})$$

Since the change only effects row k of the AP matrix, we can effectively treat all other rows as a constant.

$$p(D|\Delta A_{kl}, \sigma, AP^T) \propto \prod_j p(D_{kj}|\Delta A_{kl}, \sigma_{kj}, (AP^T)_{kj})$$

$$p(D|\Delta A_{kl}, \sigma, AP^T) \propto \prod_j \frac{1}{\sqrt{2\pi\sigma^2_{kj}}} \exp\left\{ \frac{-1}{2\sigma^2_{kj}} \left(D_{kj} - \left((AP^T)_{kj} + \Delta A_{kl} P_{jl} \right) \right)^2 \right\}$$

$$p(D|\Delta A_{kl}, \sigma, AP) \propto \exp\left\{ \sum_j \frac{-1}{2\sigma^2_{kj}} \left(D_{kj} - (AP)_{kj} - \Delta A_{kl} P_{lj} \right)^2 \right\}$$

$$p(D|\Delta A_{kl}, \sigma, AP) \propto \exp\left\{ \sum_j \frac{-P^2_{lj}}{2\sigma^2_{kj}} \left(\Delta A_{kl} - \frac{D_{kj} - (AP)_{kj}}{P_{lj}} \right)^2 \right\}$$

Similarly, for a change to P_{lk}

$$p(D|\Delta P_{lk}, \sigma, AP) \propto \exp\left\{ \sum_j \frac{-A^2_{il}}{2\sigma^2_{ij}} \left(\Delta P_{lk} - \frac{D_{ik} - (AP)_{ik}}{A_{il}} \right)^2 \right\}$$

4.2 Birth

$$p(x|D, \sigma, AP) = \prod_i \prod_j p(D_{ij}|x, \sigma_{ij}, (AP)_{ij}) p(x) \propto \prod_j p(D_{kj}|x, \sigma_{kj}, (AP)_{kj}) p(x)$$

$$p(\Delta A_{kl} = x|D, AP^T, P, \sigma) \propto \exp\left\{ \sum_j \frac{-P^2_{lj}}{2\sigma^2_{kj}} \left(x - \frac{D_{kj} - (AP)_{kj}}{P_{lj}} \right)^2 \right\} \exp\left\{ -\lambda x \right\}$$

Define $s_{k,l,j} = \frac{P_{lj}^2}{2\sigma_{kj}^2}$ and $\mu_{k,l,j} = \frac{D_{kj} - (AP)_{kj}}{P_{lj}}$

$$p(x|D,\sigma,AP) \propto \exp\Big\{ -\sum_j s_{k,l,j}\Big(x - \mu_{k,l,j}\Big)^2\Big\}\exp\Big\{-\lambda x\Big\}$$

$$p(x|D,\sigma,AP) \propto \exp\Big\{ -\sum_j s_{k,l,j}\Big(x^2 - 2x\mu_{k,l,j} + \mu_{k,l,j}^2\Big)\Big\}\exp\Big\{-\lambda x\Big\}$$

$$p(x|D,\sigma,AP) \propto \exp\Big\{ -\sum_j s_{k,l,j}\Big(x^2 - 2x\mu_{k,l,j}\Big)\Big\}\exp\Big\{-\lambda x\Big\}$$

$$p(x|D,\sigma,AP) \propto \exp\Big\{ -\lambda x - \sum_j s_{k,l,j}\Big(x^2 - 2x\mu_{k,l,j}\Big)\Big\}$$

$$p(x|D,\sigma,AP) \propto \exp\Big\{ -\lambda x - x^2\sum_j s_{k,l,j} + 2x\sum_j s_{k,l,j}\mu_{k,l,j}\Big\}$$

$$p(x|D,\sigma,AP) \propto \exp\Big\{ -x^2\sum_j s_{k,l,j} + 2x\Big(\sum_j s_{k,l,j}\mu_{k,l,j} - \lambda/2\Big)\Big\}$$

$$p(x|D,\sigma,AP) \propto \exp\Big\{\Big(-\sum_j s_{k,l,j}\Big)\Big(x^2 - 2x\Big(\frac{\sum_j s_{k,l,j}\mu_{k,l,j} - \lambda/2}{\sum_j s_{k,l,j}}\Big)\Big)\Big\}$$

$$x|D,\sigma,AP \sim \mathcal{TN}\Big(\frac{2\sum_j s_{k,l,j}\mu_{k,l,j} - \lambda}{2\sum_j s_{k,l,j}}, \frac{1}{\sqrt{2\sum_j s_{k,l,j}}}\Big)$$

We then define $s_{k,l}^{(A)} = \sum_j \frac{P_{lj}^2}{2\sigma_{kj}^2}$ and $s\mu_{k,l}^{(A)} = \sum_j \frac{P_{lj}(D_{kj} - (AP)_{kj})}{2\sigma_{kj}^2}$ so that

$$x|D,\sigma,AP \sim \mathcal{TN}\Big(\frac{2s\mu_{k,l}^{(A)} - \lambda}{2s_{k,l}^{(A)}}, \frac{1}{2\sqrt{s_{k,l}^{(A)}}}\Big)$$

If the change happens instead in the P matrix, i.e. $\Delta P_{lk} = x$ then we define

$s_{k,l}^{(P)} = \sum_i \frac{A_{il}^2}{2\sigma_{ik}^2}$ and $s\mu_{k,l}^{(P)} = \sum_i \frac{A_{il}(D_{ik} - (AP)_{ik})}{2\sigma_{ik}^2}$ so that

$$x|D,\sigma,AP \sim \mathcal{TN}\Big(\frac{2s\mu_{k,l}^{(P)} - \lambda}{2s_{k,l}^{(P)}}, \frac{1}{2\sqrt{s_{k,l}^{(P)}}}\Big)$$

4.3 Death

4.3.1 Gibbs Sampling

$$p(x|x_0,D,\sigma,AP) \propto \prod_j \exp\Big\{\frac{1}{2\sigma_{kj}^2}\Big(D_{kj} - (AP)_{kj} - (x - x_0)P_{lj}\Big)^2\Big\}\exp\Big\{-\lambda x\Big\}$$

$$p(x|x_0,D,\sigma,AP) \propto \exp\Big\{\sum_j \frac{-P_{lj}}{2\sigma_{kj}^2}\Big(x - \frac{x_0P_{lj} + D_{kj} - (AP)_{kj}}{P_{lj}}\Big)^2\Big\}\exp\Big\{-\lambda x\Big\}$$

So now if we define $s_{k,l,j} = \frac{P_{lj}^2}{2\sigma_{kj}^2}$ and $\mu_{k,l,j} = \frac{D_{kj} - (AP)_{kj} + x_0P_{lj}}{P_{lj}}$ the results are the same as the birth section, indeed if $X_0 = 0$ we have exactly a birth step (without the additional accept/reject we're about to do with this new atom).

4.3.2 Likelihood Calculation

$\mathcal{L}(x,x_0) = \log(p(D|x_0,x,\sigma,AP)) - \log(p(D|-x_0,\sigma,AP))$

$$\mathcal{L}(x,x_0) = \sum_j \frac{-P_{lj}^2}{2\sigma_{kj}^2}\Big(x - x_0 - \frac{D_{kj} - (AP)_{kj}}{P_{lj}}\Big)^2 - \sum_j \frac{-P_{lj}^2}{2\sigma_{kj}^2}\Big(-x_0 - \frac{D_{kj} - (AP)_{kj}}{P_{lj}}\Big)^2$$

$$\mathcal{L}(x,x_0) = \sum_j \frac{-P_{lj}^2}{2\sigma_{kj}^2}\left(\left(x - x_0 - \frac{D_{kj} - (AP)_{kj}}{P_{lj}}\right)^2 - \left(-x_0 - \frac{D_{kj} - (AP)_{kj}}{P_{lj}}\right)^2\right)$$

define: $a_j = \dfrac{D_{kj} - (AP)_{kj}}{P_{lj}}$

$$\mathcal{L}(x,x_0) = \sum_j \frac{-P_{lj}^2}{2\sigma_{kj}^2}\left((x - x_0 - a_j)^2 - (-x_0 - a_j)^2\right)$$

$$\mathcal{L}(x,x_0) = \sum_j \frac{-P_{lj}^2}{2\sigma_{kj}^2}\left(x^2 - xx_0 - xa_j - x_0x + x_0^2 + x_0a_j - a_jx + a_jx_0 + a_j^2 - x_0^2 - 2x_0a_j - a_j^2\right)$$

$$\mathcal{L}(x,x_0) = \sum_j \frac{-P_{lj}^2}{2\sigma_{kj}^2}\left(x^2 - 2x_0x - 2a_jx\right)$$

$$\mathcal{L}(x,x_0) = \sum_j \frac{-xP_{lj}^2}{2\sigma_{kj}^2}\left(x - 2(x_0 + a_j)\right)$$

$$\mathcal{L}(x,x_0) = \sum_j \frac{-xP_{lj}^2}{2\sigma_{kj}^2}\left(x - \frac{2x_0P_{lj} + 2D_{kj} - 2(AP)_{kj}}{P_{lj}}\right)$$

$$\mathcal{L}(x,x_0) = \sum_j \frac{-xP_{lj}^2}{2\sigma_{kj}^2}\left(x - \frac{2x_0P_{lj} + 2D_{kj} - 2(AP)_{kj}}{P_{lj}}\right)$$

4.4 Move

4.5 Exchange

4.6 Birth/Death Relative Probability

4.7 Sparse Matrix Calculations

4.8 Sparse χ^2 Calculation