Lineage-based identification of cellular states and expression programs

Tatsunori Hashimoto, Tommi Jaakkola, Richard Sherwood, Esteban O. Mazzoni, Hynek Wichterle and David Gifford

Department of Computer Science and Electrical Engineering, Massachusetts Institute of Technology, Cambridge, MA 02139
Brigham Women’s Hospital and Harvard Medical School, Boston, MA 02115 and Departments of Pathology, Neurology and Neuroscience, Center for Motor Neuron Biology and Disease and Columbia Stem Cell Initiative, Columbia University Medical Center, New York, NY 10032, USA

ABSTRACT

Summary: We present a method, LineageProgram, that uses the developmental lineage relationship of observed gene expression measurements to improve the learning of developmentally relevant cellular states and expression programs. We find that incorporating lineage information allows us to significantly improve both the predictive power and interpretability of expression programs that are derived from expression measurements from in vitro differentiation experiments. The lineage tree of a differentiation experiment is a tree whose nodes describe all of the unique expression states in the input expression measurements, and edges describe the experimental perturbations applied to cells. Our method, LineageProgram, is based on a log-linear model with parameters that reflect changes along the lineage tree. Regularization with $L_1$ that based methods controls the parameters in three distinct ways: the number of genes change between two cellular states, the number of unique cellular states, and the number of underlying factors responsible for changes in cell state. The model is estimated with proximal operators to quickly discover a small number of key cell states and gene sets. Comparisons with existing factorization, techniques, such as singular value decomposition and non-negative matrix factorization show that our method provides higher predictive power in held, out tests while inducing sparse and biologically relevant gene sets.

Contact: gifford@mit.edu

1 INTRODUCTION

The directed differentiation of embryonic stem (ES) cells into therapeutically important cell types holds great potential for regenerative medicine. Identifying stage-specific transcription factor candidates for the directed differentiation of ES cells has been difficult, and the computational identification of lineage-associated transcription factor programs would significantly benefit this process.

LineageProgram is a new method that identifies the experimental protocols that result in the same cellular states (Figure 1). It further decomposes these states into interpretable expression programs, which we define to be sets of co-varying genes. In contrast to analyzing the correlation between genes, we define a program to be sets of genes that co-vary during a differentiation event. Analysis of developmental expression data has revealed the existence of expression programs regulating pluripotency across many cell types [9] as well as lineage-specific programs. We provide a principled method that discovers both types of programs. LineageProgram is a log-linear model that uses latent factors and $L_1$ regularization to obtain sparse parameters, structures and spectra.

Our primary goal is to estimate expression programs that are informed and improved by lineage information. To our knowledge, LineageProgram is the first method to approach this problem. Prior work in the estimation of developmental expression programs has used biclustering, factor or topic decompositions, mixture models and self organizing maps, all of which simultaneously identify expression programs and sample clusters, but without incorporating information from the experimental lineage. As shown in studies of expression time series, treating dependent expression data as independent observations can lead to significant loss of information [3]. Separate prior work has focused on estimating lineage trees and lineage states in the absence of expression programs. Differentiation has been modeled both as a time series without branching and de novo tree estimation.

The remainder of this article presents the LineageProgram model (Section 2), a comparative analysis of LineageProgram with other methods on lineage-associated expression data from motor neuron and pancreatic development (Section 3) and a conclusion about what we have learned about using lineage information (Section 4).
2 THE LINEAGEPROGRAM METHOD

Model

LineageProgram operates on $N$ expression measurements of $P$ genes made on an experimental tree with $M$ nodes, where the root corresponds to the ES cell state and edges correspond to experimental interventions. We represent the cell state at a node $i$ with the probability vector $\theta_i$, whose $k$th component is the probability that gene $k$ is transcribed. The expression measurement is modeled as proportional to a multinomial draw from $\theta_i$. Methods such as GeneProgram [10, 16] have successfully used this discrete-count model for expression. Our objective function is the continuous extension of the multinomial likelihood function. We will show that this natural continuous extension exists as a discretization limit and allows us to handle continuous data such as microarray measurements directly.

A differentiation event is a change in $\theta_i$, which we represent by a log-odds change $\eta_i$. The change of a gene $k$ from a parent state $i$ with vector $\theta_i$ to child state $j$ is written as

$$\eta_{ij,k} = \frac{\theta_{ij,k}\exp(\theta_{ij,k})}{\sum_l \theta_{ij,l}\exp(\theta_{ij,l})}$$

This formulation of a log-odds count model has been shown to outperform analogous Latent Dirichlet Allocation type models [8].

We represent the root stem cell state in the experimental tree as a log probability vector $\phi$ of size $P$, and the remaining states are represented as log odds-changes from their parent. For each node on the cell state tree, we write the expression probability as the sum of log changes along its path from the ES state and the ES cell expression $\phi$. Let $T_i$ be the set of nodes along the path from node $j$ to the ES state. Then the probability of observing gene $k$ at node $j$ is given as follows:

$$\theta_{j,k} = \frac{\theta_{i,k}\exp(\theta_{i,k})}{\sum_l \theta_{i,l}\exp(\theta_{i,l})}$$

We represent the experimental structure as two matrix multiplications: an $M \times M$ path sum matrix $T$ with $T_{ij} = 1$ if $i \in T_j$ and zero otherwise and an $N \times M$ observation matrix $D$ with $D_{ij} = 1$, if the $i$th expression measurement was made at node $j$ and zero otherwise.

The parameters are represented as two matrices, $M \times P$ parameter matrix $\eta$ (which we will regularize to be sparse and low-rank) and a $1 \times P$ ES expression vector $\phi$. Given the $N \times P$ data matrix $X_{t,i,k}$, which we interpret to be proportional to counts of an expression event, our log likelihood takes the form

$$llh(\eta) = \sum_{t,i,k} X_{t,i,k} \log \left( \frac{\exp((DT_\eta)_{ik} + \phi_i)}{\sum_k \exp((DT_\eta)_{ik} + \phi_k)} \right)$$

We design our regularization with three objectives: there should be few genes changing per differentiation event (L1 penalty on $\eta$), few unique differentiation events (L2 penalty on the rows of $\eta$) and few programs needed to explain the lineage (trace norm penalty on $\eta$). The L1 penalty is the sum of absolute values of $\eta$, which induces $\eta$ to have entries with zeroes. The L2 penalty is the sum of the row-norms of $\eta$, which induces $\eta$ to have rows that are all zeroes. Finally, the trace norm penalty is the sum of the singular values of $\eta$, which induces $\eta$ to have low rank.

With the regularization and defining the shorthand notation $|\eta|_1 = \sum_l |\eta_{lk}|$ and $|\eta|_2 = (\sum_l |\eta_{lk}|^2)^{1/2}$, the overall objective $f$ takes the form

$$f(\eta) = -llh(\eta) + \lambda_1 |\eta|_1 + \lambda_2 |\eta|_2 + \lambda_3 ||\eta||_T$$

We show that discretizing the data, $X$, to some precision $\beta$, and taking the limit as $\beta \to \infty$ has the equivalent minima (up to scaling and the zero set) by using $X$ without discretization. Note that the gradient of the log-likelihood function takes the form

$$\lim_{\beta \to 0} \beta \sum_{t,i,k} (X_{t,i,k}/\beta) \left( \frac{\exp((DT_\eta)_{ik} + \phi_i)}{\sum_k \exp((DT_\eta)_{ik} + \phi_k)} - \frac{1}{\beta} \sum_k X_{t,i,k} \frac{\exp((DT_\eta)_{ik} + \phi_i)}{\sum_k \exp((DT_\eta)_{ik} + \phi_k)} \right)$$

which is the continuous extension of $llh$ up to a constant $1/\beta$. The regularization terms $|\eta|_1$, $|\eta|_2$, and $|\eta||_T$ are convex, but not strictly convex, so the optima of the continuous extension and the limit can differ up to elements of the zero set. Testing both small discretization and the continuous extension, we find no difference in results, but for completeness we use a threshold of $\epsilon = 5 \times 10^{-6}$ to set a small neighborhood near zero to be part of the zero set.

Finally, we define the concept of an expression program as a set of basis vectors spanning $\eta$. The trace norm regularization implicitly penalizes the rank of matrix $\eta$, and for large $\lambda_3$, $\eta$ will have small rank and can be represented as the linear combination of a few “basis” programs. We choose the singular value decomposition of $T_\eta$ as our program decomposition. The first $k$ programs have a natural interpretation as the best rank-$k$ approximation of the unnormalized log expression parameters.

Inference

The advantage of our method over topic model formulations is the convexity of our objective $f$, which guarantees fast convergence to the global maxima. Our overall inference strategy is to use gradient descent on the likelihood combined with proximal steps on each of the regularization terms. To speed convergence, we also use the accelerated proximal gradient method. The proximal gradient method allows us to efficiently optimize convex functions of the form $f = f^* + g$, where $f^*$ is convex differentiable and $g$ is convex and continuous. $f$ is optimized with a gradient step on $f^*$ followed by a proximal operator, which uses a quadratic approximation of $f^*$ to optimize $f^* + g$. Given $x_t$, we generate the next iterate $x_{t+1}$ with the following update

$$x_{t+1} = \text{Prox}_{g_t}(x_t - \epsilon \nabla f(x_t))$$

$$\text{Prox}_{g_t}(x_t) = \text{argmin}_Y \|x_t - Y\|_2^2 + \epsilon g(Y)$$
Fig. 2. The goodness of fit as measured by held-out likelihood shows that the optimal model has few of edges and programs.

Fig. 3. Cross-validated mean square errors on the full cell tree shows good predictive performance by LineageProgram.

In our case, our log likelihood is concave differentiable, and there are three convex continuous functions: $g_1$, $g_2$, and $g_3$, corresponding to each regularization term.

$$ f^*(\eta) = f(\eta) + \lambda_1 \sum_i |\eta_i| + \lambda_2 \sum_i ||\eta_i||$$

The gradient for $f^*$ is given by the difference between predicted and observed counts.

$$ \nabla f^*(\eta)_{jk} = \sum_t X(t, k) - \left( \sum_{l} X(t, l) \exp((D\eta)_l + \phi_l) \right) / \sum_{l} \exp((D\eta)_l + \phi_l) $$

The proximal operators for $g_1$ and $g_2$ are the soft-threshold operators.
Prox$_{\epsilon}^g(i, k) = \frac{\| i \|_2}{\max(\lambda_1 \epsilon, 0)} \begin{cases} i_k = \epsilon \lambda_2 & \text{if } \| i \|_2 > \epsilon \lambda_2 \\ 0 & \text{otherwise} \end{cases}

For $\epsilon$, the proximal operator can be written in closed form in terms of its singular value decomposition (SVD) [14]. Let $\eta = UDV^T$ and $\max(D - \lambda V, 0)$ be the SVD and entrywise subtraction followed by thresholding, then the proximal operator takes the form

$$\text{Prox}_{\epsilon}^g(\eta) = U \max(D - \lambda V, 0)V^T.$$

At each step of the optimizer, we take some $\epsilon$ and step size $\eta$ and produce the next iterate with

$$x_{t+1} = \text{Prox}_{\epsilon}^g(x_t - \eta \nabla f(x_t)).$$

The sequential proximal gradient converges for our objective due to separability. We also make use of the accelerated gradient method by which finds a sequence of $\epsilon_t$ which converges toward the optima, using an internal variable $\gamma_t$ and a magnification of the gradient, $\alpha_t$ to increase convergence rates near the mode.

$$x_{t+1} = \text{Prox}_{\epsilon}^g(\text{Prox}_{\epsilon}^{g_2}((x_t - \gamma_t \nabla f(x_t))))$$

$$\eta_{t+1} = \frac{1 + \sqrt{4 \gamma_t^2 + 1}}{2} \gamma_t,$$

$$y_{t+1} = x_t + \frac{\eta_{t+1}}{E_t} (x_t - x_{t-1}).$$

In the context of a single proximal operator, this produces the optimal quadratic first-order convergence rate. In our case, the multiple proximal operators do not provide a guaranteed convergence rate, but in practice, we find the accelerated gradient makes convergence significantly faster. An implementation of this inference method as well as the results of our analysis are available from our website at http://psrg.csail.mit.edu/resources.html.

Inference with this method is fast enough that we are able to fit the model across a 50 × 50 × 50 grid of all valid $\lambda_1, \lambda_2$ and $\lambda_3$, which we use to set the regularization parameters as described in the next section.

### 3 RESULTS

The algorithm was tested on directed differentiation experiments for murine pancreatic progenitors and motor neurons, as shown in Figure 4. Both lines were produced using known differentiation protocols. The pancreatic line has a large number of states, but relatively few replicates, while the motor neuron line has multiple replicates per state. The 88 microarray measurements of the pancreatic line were performed with Illumina bead arrays, while the 17 in the motor neuron line were performed with Affymetrix 430a2 microarrays. We rank-match the Illumina data to the Affymetrix data in order to reduce the data to the same scale.

The quantile normalization technique is described in further detail by Irizarry et al. (2003).

### Cross-validation procedure

We use cross-validation to estimate the regularization parameters $\lambda_1$, $\lambda_2$ and $\lambda_3$ over the 50 × 50 × 50 grid of all nondegenerate values. For every experimental state with more than one observation, we include one observation in the training set and include the rest in the held-out set. We choose to leave out observations per-node rather than over all observations, since we must leave at least one observation at each node in order to make a prediction at the node. The performance of the model is measured in terms of held-out likelihood, which indicates goodness of fit, and squared error, measures predictive performance.

Cross-validation results in Figure 5 show the existence of a sharp drop in predictive power around a dozen edges and programs; this sharp transition suggests that there exists a necessary level of regularization for our algorithm to generalize well.
Fig. 6. Gene set enrichment in Gene Ontology (GO) and manually curated developmental gene sets show programs recovered by LineageProgram are biologically relevant.

The optimal regularization parameters discovered by cross-validation significantly penalize both the number of edges and rank of the parameter $\eta$. At this optimal value, we find that the estimated cell state tree in Figure 8 is significantly sparser than the experimental tree. We also found that a strong $L_2$ penalty dominated the group $L_2$ penalties, resulting in $\lambda_2$ controlling both structural and parameter sparsity.

Necessity of trace norm regularization

The trace-norm penalty is a critical part of modeling large branching factors. The $L_1$ and $L_2$ penalties shrink the changes between nodes to zero and bring the probabilities at each child-node toward their common parent. Therefore, the two edge penalties, $\lambda_1$ and $\lambda_2$, control whether leaves differ at all from their parent, rather than how they differ. In contrast, the trace norm penalty restricts directions in which the leaves can differ by forcing the leaves to lie within a small subspace near their parent.

To test the necessity of trace norm regularization, we compared our model with and without trace-norm regularization. For both models, we fit the regularization parameters through cross-validation and compared the mean squared error on a common held-out set of 16 arrays selected by removing one array from each node with replicates. Results in Figure 5 indicate that without the trace norm penalty, 7 of the 16 held-out observations show significant increases in squared error. All seven of these observations are children of the highest degree node in Figure 8 suggesting that the trace norm penalty plays a key role preventing overfitting on highly branching data.

Methods compared

Our algorithm was compared against three existing classes of approaches: SVD, non-negative matrix factorization (NMF) and latent Dirichlet allocation (LDA) on both held-out prediction error and quantitative program metrics. We were restricted to considering non-lineage-informed methods since we did not find any lineage informed methods in the literature.

For each competing method, we tested major variants of the algorithms and chose to compare only to the best variant. For SVD, we tried direct decomposition of the data, mean subtraction (resulting in Principal Component Analysis (PCA)) and mean subtraction on pancreatic and motor neuron branches; simple mean subtraction outperformed the others and is shown here. For NMF, we tested Kullback-Leibler (KL) divergence and square distance minimization objective proposed by as well as sparse NMF; we use KL divergence minimization. For LDA, we used the collapsed Gibbs sampler as well as variational Bayes updates and found the Gibbs sampler with 20000 samples to perform best.

Low cross-validation error on branching data

The held-out mean squared prediction error measures the generalization performance of each of the algorithms. We split the expression measurements into cross-validation and held-out sets selecting 16 replicate experiments one from each node with more than one measurement. The cross-validation set is split into training and test sets using the procedure described in the cross-validation section. The models are fit on the training data and we calculate the squared error between the predicted values from the model and the test data.

Training on the full set of motor neuron and pancreatic data (Figure 3a), we find that overall, LineageProgram has the lowest mean squared prediction error. On lineages with no branching, we would expect to see SVD perform best due to its direct minimization of squared error. We find that on the motor neuron lineage with no branching, our algorithm performs like SVD. In the worst case of non-branching data, our algorithm compares favorably to current factorization methods. Interestingly, LDA degenerates on the motor neuron held-out sets with behavior consistent on both training methods and across multiple replications. We find that LDA overfits on the later stage pancreatic states at the expense of the motor neuron states, which is consistent with behavior observed in prior work [8].

To rule out the possibility that the low performance of the competing algorithms was due to the inclusion of two differing array
As with prior work \[10\], we measure Gene Ontology (GO) term enrichment as a proxy for biological relevance. Finally, we want each program to encode for biologically meaningful sets of genes.

Comparison of program quality

Although methods such as SVD can reconstruct expression values well, these program decompositions are often difficult to interpret biologically. Therefore, we quantify the quality of the decomposition in terms of sparsity and biological relevance. First, we would hope for each program to have several dominant genes in order to produce candidate genes for further analysis. We measure this objective with the proportion of the program explained by the coefficient distribution of the programs. Second, we want each program to map to a unique developmental gene set. We measure this objective with the proportion of the program explained by the dominant developmental gene set.

Fewer genes are used to model the lineage

To measure sparsity, we normalize the programs from all methods as a function of Benjamini Hochberg corrected \(p\)-values (Figure 5), we find that LineageProgram achieves significantly higher GO enrichments when compared to the other methods. The higher GO enrichment across all \(p\)-values suggests that the programs recovered by LineageProgram more closely match subtrees of the GO annotation than those recovered by competing methods. If this were purely a result of chance, we would expect LineageProgram to outperform the others on a small subset of \(p\)-value cutoffs, rather than across all cutoffs. In combination with our sparsity results, this suggests that LineageProgram discovers small sets of biologically relevant genes in each program.

To ensure that GO enrichment is in developmentally relevant categories, we also test the uniqueness of each program. In an ideal decomposition, each program would encode a different aspect of the developmental process. We use six developmental gene sets found in the literature, including four GO categories (‘stem cell maintenance’, ‘endocrine pancreas differentiation’, ‘embryonic skeletal development’, and ‘anterior posterior development’), as well as marker genes for motor neurons (Pax6, Mnx1, Isl1, Lhx1 and Lhx3) [13] and pancreas (Prox1, Pdx1, Hh9 and Nkx6.1) [15]. For each program, we measure the proportion of the program’s \(L_2\) norm that is accounted by genes in the most activated category. The results in Figure 6 show that LineageProgram as well as NMF have programs closely matching known differentiation programs. These correspond to the loss of pluripotency; LDA and SVD were unable to discover programs that could be mapped solely to pluripotency. Importantly, LineageProgram maintains a relatively high correspondence to known gene sets across most of its programs, indicating its ability to encode small, highly specific gene modules corresponding to both the pancreatic and motor neuron lineages.

Analysis of full lineage data

Finally, we train LineageProgram on the full dataset to estimate lineage programs and use cross-validation on replicates to estimate regularization parameters. The resulting tree is sparse, with only two branching points (Figure 8). The first branching point corresponds to the split between motor neuron and pancreatic lineages. The other branching point differentiates a successful Day 6 pre-endoderm differentiation from a late-stage Sox17 overexpression experiment.

We compared the lineage tree from the model with a manual annotation produced separately from this project. Taking only the vertices corresponding to known cellular states, we match the manual curation almost exactly, successfully reducing all the branches before the Day 4 preendoderm and only mis-merging the Day 5 endoderm with the Day 6 posterior foregut. The regularization and cross-validation have successfully pruned nearly all spurious branches of the line tree.

The Sox17 branch in Figure 8 represents a late-stage experiment aimed at recovering competency in Day 8 endoderm using Sox17 overexpression. Although the experiment shows significant expression level differences from the Day 4 pre-endoderm, we have
Identification of enriched GO terms and genes also show close correspondence with known biological facts. We correctly identify all major stages of the motor neuron differentiation process, from loss of pluripotency and development of positional identity to spinal motor neuron specification. On the pancreatic branch, we identify loss of pluripotency, suppression of mesodermal identity and finally specification of endodermal and pancreatic identity. We include full summary output of the enriched genes and GOs for each program in the appendix.

CONCLUSION

We have presented LineageProgram, a log-linear sparse regularized model and inference algorithm for lineage-associated expression data that provide strong interpretability with no loss in predictive power. Existing flat modeling methods were unable to cope with the large number of leaves that occur at the ends of the differentiation experiment, and significant statistical power is lost re-estimating the background cellular expression levels. Our biological metrics also suggest that flat factorization methods do not extract biologically meaningful expression programs, and modeling the differences between each differentiation state is necessary to estimating fine behaviors occurring during differentiation.

Our analysis of the combined pancreas and motor neuron data provides a computational analysis of the pancreatic and motor neuron differentiation pathways that recapitulate known biological markers and states. The ability of the sparse regularization to remove spurious branches in the lineage tree suggests the use of this model to estimate novel cellular states in later-stage differentiation experiments, where explicit cellular states are less characterized.

Although we have restricted our model to expression-based data, we hope to extend this framework of $L_1$-regularized branching models to sequencing data with similar structure. The penalty model presented here would allow a wide variety of log-convex generative models to incorporate lineage-associated structure.

Funding: National Institutes of Health [P01-NS055923 and 1-UL1-RR024920 to D.K.G.].

Conflict of Interest: none declared.

REFERENCES

Cellular states and expression programs

Costal., et al. (2007) Gene expression trees in lymphoid development. BMC Immunology, 8, 25.
Eisenstein,J. et al. (2011) Sparse additive generative models of text.