Learning a Constrained and Simplified Chemical System Modeled by a Bayesian Network

I implemented a hill-climbing algorithm for Bayesian Network learning and tested it on synthetic data from a simplified and constrained chemical reaction system. In this paper, I will describe the learning algorithm, the system modeled, the implementation, and the results of some experiments on the system.

I. Brief Overview of Bayesian Network Learning

A Bayesian network is specified by a directed acyclic graph (DAG) and local conditional probability tables (CPTs). It models a stochastic system with conditional independence and causal relationships. Each observable random variable is associated with a node in the DAG, and each node is associated with a CPT that maps the values (observations) of its parent nodes to probability distributions for its own observable outcomes. Learning a Bayesian network, both DAG and CPTs, involves two very different activities: learning the CPTs, or parameters, and learning the DAG structure. The decisions I have made in this project relied on class slides from this quarter in addition to Page's lecture notes (Page, 2009) and sections of Heckerman's tutorial (Heckerman, 1995).

Learning the structure can be formulated as a search problem. A proposed structure S^h can be seen as a state. It can be modified, as noted on Page's notes, by a transition function that removes, flips, or adds a single edge. Page's notes also point out that in most cases greedy hill-climbing search is acceptable, though other approaches remain open topics of current research. My implementation uses hill-climbing with multiple climbers (a parameter in the implementation) to attempt to avoid getting stuck in local maxima. One of the hill-climbers always begins with an empty structure, and all other climbers start with a randomized DAG. These structures are formed by shuffling all possible edges and then adding, in order, those that produce no cycles.

Both Page and Heckerman identify a common evaluation function, the Bayesian Information Criterion (BIC):

$$\log p\left(D|S^h\right) \approx \log p\left(D|\widehat{\theta_s}, S^h\right) - \frac{d}{2}\log N$$

Here, D is the data used for learning. θ_s and d are the learned parameters for the given structure, and their complexity, respectively. The parameters in the CPTs can be found given S^h by calculating posterior probabilities from the provided data. d can be found by adding the sizes of the CPTs for each of the nodes. The first term, $\log p\left(D|\widehat{\theta_s}, S^h\right)$, is the log of the probability of this data given the structure and the learned parameters, or CPTs. This is found by adding together the logs of the probabilities of the proposed structure producing each of the samples in the data. The second term, $d/2 \log N$, punishes the complexity of the structure.

A Bayesian network reflects complex dependencies and relationships, or simplified beliefs about those relationships, through structure and conditional probabilities. Sampling from that structure provides data that has an interpretation sensitive to those relationships. Using the sampled data to create a structure is certainly possible without a meaningful model underlying the original network. However, there are very many ways to "generate random networks." Arbitrary choices made in generating data manifest in the algorithm's performance. For example, in the data from the chemical system defined in the next section, the amount of chemical product is not sensitive to an excess of only one of the necessary reactants. Dependence between reactants might be learned that is justifiable, but redundant. In practice, punishing structures for complexity of the parameter space prevents the formation of these extraneous edges *in addition to* causing slightly worse overall learning (see test 1 in section IV), which makes sense in this particular application, but not necessarily in others.

II. Simplified Chemical System: The Beaker

To generate synthetic data from which to learn, I have defined a simplified chemical system that models a beaker filled with chemicals, interacting with one another through constrained chemical reactions, but with stochastic results. The system is specified by *nchem*, *nval*, *heat* and a list of reactions ordered by priority. Reactions describe how many moles (effectively, units) of each *reactant* chemical are needed to produce a certain amount of each *product* chemical. Results of interactions within it, in the form of observed quantities of chemicals, can be modeled using a Bayesian network. Each chemical is associated with a random variable, C_i for $i{:}0 \rightarrow$ *nchem*,

with integer outcomes between 0 and *nval* inclusively. These outcomes describe how much of a chemical is observed in moles. The *heat* parameter, together with the reactions involved, informs the probability distributions of these outcomes. Modeling this system asynchronously requires some constraints:

- *The product-once constraint*: a chemical may appear in at most one reaction as a product.
- *The product-reactant duality constraint*: for every chemical, there is an identifiable set of *enabling* chemicals – those that are reactants in the equation which produces it – and a distinct set of *inhibiting* chemicals – those that are produced by the enabling chemicals in higher-priority reactions.

The observations are not time-sensitive. For example, in the reaction *2A + B > 3C*, if 6 moles of *C* are observed ($C_C = 6$), then 4 moles of *A* must have been used up. The outcome of C_A, the random variable associated with *A*, is not updated to reflect the loss of 4 moles; if we had previously observed there to be 8 moles of *A*, this would still be the value of C_A. To determine how much *A* there is after it is involved in reactions, define the function *usage(A,N,X)*, or *usage of A due to N moles of X*. This function always returns 0 if *X* is not a direct product of *A* in some reaction. By the *product-once constraint*, it is guaranteed that if *X* is a product of *A*, then all of *X* came from the reaction involving *A*. Then, the answer to "*how much A is in this beaker?"* is given by:

$$\boldsymbol{C_A} - \sum_{\boldsymbol{observed\ C_X}} \boldsymbol{usage}(\boldsymbol{A}, \boldsymbol{C_X}, \boldsymbol{X}) = C_A - usage(A, C_C, C) = 8 - usage(A, 6, C) = 8 - 4 = 4$$

Suppose, in addition to the reaction *2A + B > 3C* there is another reaction possible, *A+B > 2D*. If C_A and C_B have been observed, what is the probability distribution of C_C? First, we impose a *priority ordering* on the equations. If *A + B > 2D* has a higher priority than *2A+B>3C*, it means that the amount of *D* produced will affect the amount of *C* produced, but not vice versa. How much *A* is available for *C* to use is thus given by C_A – *usage(A,*C_D*,D)*. Having calculating the available amounts of both *A* and *B*, we use the proportions given by the equations to find how much *C* each can be produced. One of *A* or *B* might be able to produce less *C* that the other; this *limiting reactant* dictates the maximum possible amount of *C*. This amount may not exceed *nval*.

In this stochastic system, the amount of product produced by a given reaction is not always the maximum possible, as determined by the above calculations. The products of a given reaction are independent: *e.g.*, *A+B>C+D* may yield different amounts of C and D. *Heat* determines the tendency of the reaction to go to completion with respect to a given product. There is always a non-zero probability that an amount between 0 and the maximum possible of each product will be produced. At *heat=0.5*, this is a uniform distribution. In general, for *heat* between *0* and *1*, the lowest-probability outcome is assigned *1*, and the distribution forms a line with a *0.5-heat* slope. Then this is normalized to form a probability distribution. For example, if the maximum amount of a product is *3* moles, and *heat=1*, then *p(0 moles)=0.1*; *p(1 mole)=0.2*; *p(2 moles)=0.3*; and *p(3 moles)=0.4*

How Many Samples? The synthetic data tended to require incredibly different – sometimes by orders of magnitude – sample sizes depending on small tweaks to the chemical system's parameters. As a result, I implemented a method that uses a parameterized data consistency heuristic to determine the smallest appropriate number of samples. I used data *consistency*, defined here as a function of a Bayesian network, a set of samples *S* and a number of tests $t \geq 1$. *t* samples, each of size |*S*|, are compared in distribution *S*. This comparison treats each pair of sets as histograms, with outcome vectors mapped to then number of times they were observed in the set. It sums, for each outcome vector that occurred in *either* set, the minimum number of observations. This sum is divided by |S|. The results of *t* comparisons are averaged to produce a metric, between 0 and 1, for how consistent the given sample set is to other sets drawn from the same network. This metric is used as a heuristic to summarize how "typical" a dataset is to the output of a particular network.

III. Java Implementation

Detailed documentation and the entire source code can be found in the submitted archive. The experiments discussed in the next section use BKR.jar, whose parameters are described in Table 1 on the next page. The application reads as input a list of chemical system structures, specified in the format illustrated in Appendix B. If more than one structure is provided, the batch parameter will be used to run that many tests on each structure,

resetting the random-number generator every time with the specified seed. The output includes data about what initial setting were; learner performance output for each run (see Table 2); and, at the end of the file, a verbose printout of original structures and conditional probabilities, and learned structures in every iteration. An example call to run an experiment: **java -jar dist/BKR.jar -gibbs -complex -lsamp 0.85 -heat 1 -batch 25 > 56tests-g-c-85-1.out < 56tests.txt**

Name	Type	Default value	Description	Parameterizes the…
complex	*Boolean*	FALSE	punish structure complexity in evaluation function	… search algorithm
nsearch	*Integer*	5	number of simultaneous greedy searchers (in addition to blank-starting one)	
heat	*Double*	0.5	the heat constant (between 0 and 1)	… chemical system
nval	*Integer*	4	the maximum amount of any chemical in this system	
gibbs	*Boolean*	FALSE	use Gibbs sampling (as opposed to starting from scratch on each sample)	… how/how much to sample from a given network
Nsamp	*Integer*	-1	number of samples (-1 means find this amount dynamically)	
Lsamp	*Double*	0.9	limit the number of samples to the min. that achieves this data consistency	
Nsim	*Integer*	5	number of datasets that are compared to a given dataset to find consistency	
Seed	*Integer*	1500	seed the random-number generator with this	… running of a batch of tests
Batch	*Integer*	1	run the specified number of times with same configuration	

Table 1: *Application parameters (also available by calling the application with the flag* **-help***)*

IV. Experimental Results

I ran a series of tests where I varied parameters and structures learned. Most runs described were completed on the order of a few second or minutes, but some more complicated structures required hours. I ran many more tests than described here. The structures I focused on were chain, cross, and tree (examples of different sizes are in Appendix A). I found that trees were, remarkably, almost always learned perfectly, even on inconsistent data, though they also tended to take longer: tree15, the largest, took >12 hours to complete 30 trials. I expected the cross structure to be difficult to learn, but it seemed even trickier than I intended. The chain structure was, as expected, quick and correct.

# samples	The number of samples taken from the original network
consistency$_{in}$	The *consistency* (see end of section II) given the nsim parameter (see Table 1) and the original network
# updates	The number of updates before no learner could find a better solution
score	The score (found by evaluation function) of the best (and final) network the greedy searcher(s) found
consistency$_{out}$	The *consistency* given the nsim parameter and the learned network
# flipped	The number of edges in the learned DAG that exist as flipped (relative to the original)
# missing	The number of edges that are not in the learned DAG, but are in the original
# extra	The number of edges that are the in the learned DAG, but not the original

Table 2: *Data available for every run in a batch*

Test 1: *Punishing complexity in the evaluation function reduces learning*

For the structures listed in Appendix A, excluding the larger tree15, crossN with $N>5$, and chainN with $N>6$, I ran 25 trials with the complex flag and 25 without; all other parameters were left at default. With surprising consistency (see Figure 1 in Appendix B), the absence of the complex flag resulted in increased $consistency_{out}$, though an unaffected *score*. On average, $consistency_{out}$ was .92 for complex=*true*, and .935 for c=*false*. Note re-seeding the random number generator means that $consistency_{in}$ was the same in for matching trials across the two cases. However, the complex flag makes tests run a lot faster, and as the *score* was unaffected and the effects on the slightly dubious metric $consistency_{out}$ were small, the complex flag was used in subsequent tests.

Test 2: *Adding unconstrained random variables reduces* consistency$_{in}$ *and* consistency$_{out}$*, but not accuracy*

Thinking that a less constrained system would require more data to be well-represented, and would be harder to learn, I ran confusioncrossN (for $N<8$) with the complex flag and all other parameters at default. As Table 3 (Appendix B) shows, the larger number of unconstrained variables made data consistency difficult to achieve, but produced increasingly accurate structures (with less mistakes in the learned DAGs). This was bewildering. In the next test, Test 3, I wanted to check performance structures of non-inert chemicals (*i.e.*, constrained random variables) with variable $consistency_{in}$, questioning the assumption that more consistent data lead to better

learning.

Test 3: *Lower* consistency$_{in}$ *leads to worse learning*

After running 25 trials on crossN with *N<6*, and chainN with *N<7*, I was much less bewildered. I manipulated *consistency$_{in}$* by varying the *lsamp* parameter, and plotted in Figure 2 (Appendix B) the effect of changing training data consistency on the average proportion of edges learned. Proportion, rather than count, was used to combine data from different-size structures, while separating chain and cross structures, as the cross structure is so much harder to learn than the chain. The figure shows a general tendency for more edges correctly learned as the consistency increases. This is comforting, although I did expect it to be more dramatic.

Test 4: *Varying the* heat *parameter does not have a dramatic impact on learning*

Initially, I thought that as heat moved away from the middle (0.5), the system would be more constrained, and easier to learn. The more likely a reaction was to go to completion, the more reactant would be used up and unavailable for competing, but lower-priority reactions. Likewise, the less likely the reaction was to go to completion, the less product would be produced, and as the effect trickled down, the less outcome would be possible for descendants. As Figure 3 in Appendix B shows, this was not supported. *consistency$_{out}$* and *score* both do not seem to vary much with heat, and neither does accuracy of the learned structure or even the number of hill-climbing steps required to learn the system. There is an effect, however, on the number of samples required to reach the needed consistency (in this case, the default 0.9): it increases noticeably with increasing heat.

V. Conclusion

There are many things I could have done differently in the implementation. I did not implement anything to inform how likely a given structure was given knowledge of the system modeled. It could be possible to raise the score for structures that could actually represent a chemical system, versus ones that could not for obvious constraint violations. However, modifying a single edge as transition makes me doubt the goodness of this idea: these small changes would not allow leaps from between valid structures that are quite far apart in this space. It would be interesting to substitute fixed-depth tree search for hill climbing *in addition* to using various approaches for probabilistically scoring structures. This could make learning more successful.

A more notable shortcoming was the lack of a grasp of what is means for learning to be "successful" in the first place. My use of the consistency metrics to compare learning performance feels akin to licking a finger and holding it out of a window to test if the sun is out, but they were seductively easy to interpret. In retrospect, *consistency$_{out}$* should have reflected the test of the "consistency" of some held-out data, not the training data. Painfully, this would require modifying 1 line of code[1], but re-running the experiments would take another day. Initially, I thought having high consistency was vital to having data "good enough to train on," and with high consistency held-out data should look basically like training data. This failure to think things through made the experiment "analysis" in the end fairly unsatisfying.

There are also some deeply problematic aspects of the chemical system. The most egregious assumption is that products can be produced in different proportions. A different model of the chemical system would relieve it of this burden, and perhaps some others. For example, associating a random variable with each chemical *and* with each equation would be capable of modeling a system free of the offending assumption. However, it would severely constrain, in terms of space and time requirements, the sizes of systems that could be modeled.

Nevertheless, the implementation of the learning algorithm, as well as the chemical simulation, is interesting to run tests on and could be adapted to address all these problems in the future.

[1] In fact, this is available as "consistency held out" in the submitted program, side by side with the original consistency$_{out}$

Works Cited

Heckerman, D. (1995). *A tutorial on learning with Bayesian networks.* Microsoft Research.

Page, D. (2009). *CS 731: Advanced methods in artificial intelligence, with biomedical applications*. Retrieved 12 12, 2010, from http://pages.cs.wisc.edu/~dpage/cs731/

Appendix A: Chemical Structures Tested

A chemical system, as I've defined it, is specified fully by the number of chemicals present (or possible), *nchem*; a list of reactions governing them; *nval*; and *heat*. The first two can be specified as follows:

4	*nchem*	
2 0 + 1 1 > 1 2	2 moles of 0 and 1 mole of 1 yield max. 1 mole of 2	*higher priority equation*
1 0 + 2 1 > 2 3	1 mole of 0 and 2 moles of 1 yield max. 2 moles of 3	*lower priority equation*

The following is a list of structures, in the format described above, that are referred to in section IV.

"5A"	"5B"	"5C"	"chainN" (N≥2)
5 1 0 + 1 1 > 1 3 1 2 + 1 3 > 1 4	5 1 0 + 2 1 > 1 3 2 2 + 1 3 > 1 4	5 1 0 + 2 1 > 1 3 1 2 + 2 3 > 1 4	N 1 0 > 1 1 … 1 N-1 > 1 N
"6A"	**"6B"**	**"6C"**	**"confusecrossN" (N≥4)**
6 1 0 + 1 1 + 1 3 > 1 2 1 3 > 1 4 1 2 + 1 4 > 1 5	6 1 0 + 2 1 + 1 3 > 1 2 1 3 > 2 4 1 2 + 1 4 > 1 5	6 1 0 + 2 1 + 2 3 > 1 2 1 3 > 1 4 1 2 + 2 4 > 1 5	N 1 0 + 1 1 > 1 2 1 0 + 1 1 > 1 3 *(same reactions for every N; for testing "inertness" effects)*
"cross4"	**"cross5"**	**"cross6"**	**"cross10"**
4 1 0 + 1 1 > 1 2 1 0 + 1 1 > 1 3	5 1 0 + 1 1 > 1 2 1 0 + 1 1 > 1 3 1 2 + 1 3 > 1 4	6 1 0 + 1 1 > 1 2 1 0 + 1 1 > 1 3 1 2 + 1 3 > 1 4 1 2 + 1 3 > 1 5 *And so forth until…* →	10 1 0 + 1 1 > 1 2 1 0 + 1 1 > 1 3 1 2 + 1 3 > 1 4 1 2 + 1 3 > 1 5 1 4 + 1 5 > 1 6 1 4 + 1 5 > 1 7 1 6 + 1 7 > 1 8 1 6 + 1 7 > 1 9
"tree3"	**"tree7"**	**"tree15"**	
3 1 0 + 1 1 > 1 2	7 1 0 + 1 1 > 1 2 1 3 + 1 4 > 1 5 1 2 + 1 5 > 1 6	15 1 0 + 1 1 > 1 2 1 3 + 1 4 > 1 5 1 2 + 1 5 > 1 6 1 7 + 1 8 > 1 9 1 10 + 1 11 > 1 12 1 9 + 1 12 > 1 13 1 6 + 1 13 > 1 14	

Appendix B: Results & Figures[2]

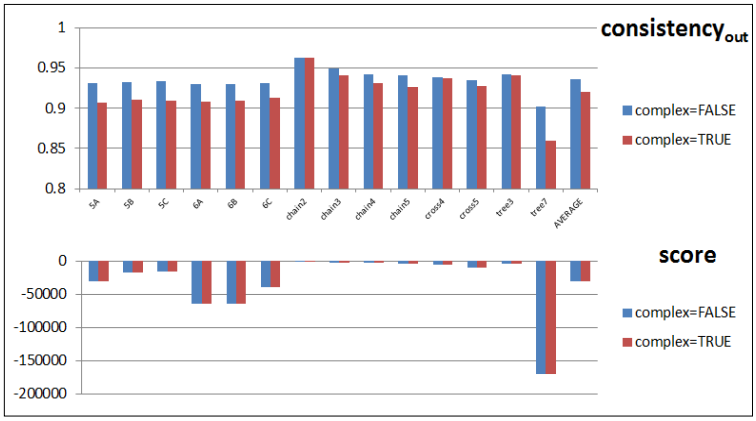

Figure 1: *Results for Test 1. Each bar represents average over 25 trials.*

Original System		Sample to Learn From		Greedy Learning Result			Mistakes in Learned DAG		
nchem	**|E|**	**# samples**	**consistency$_{in}$**	**# updates**	**score**	**consistency$_{out}$**	**# flipped**	**# missing**	**# extra**
4	5	3360	0.911119	1.16	-6168.74	0.937643	3.44	0	1
5	5	13660	0.90099	5.16	-34582.6	0.906406	3.4	0	1
6	5	50000	0.885076	6.08	-161468	0.886164	1	0	0.04
7	5	50000	0.747828	6.28	-196408	0.748841	1	0	0.12
8	5	50000	0.502366	6.08	-231360	0.502479	1	0	0.04

Table 3: *Results for Test 2. Each row represents the* average *over 25 trials for that structure.*

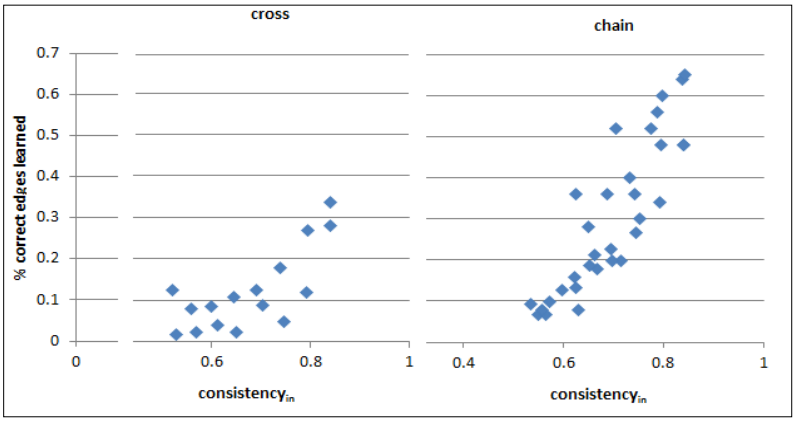

Figure 2: *Results for Test 3. Each point is an average over 25 trials.*

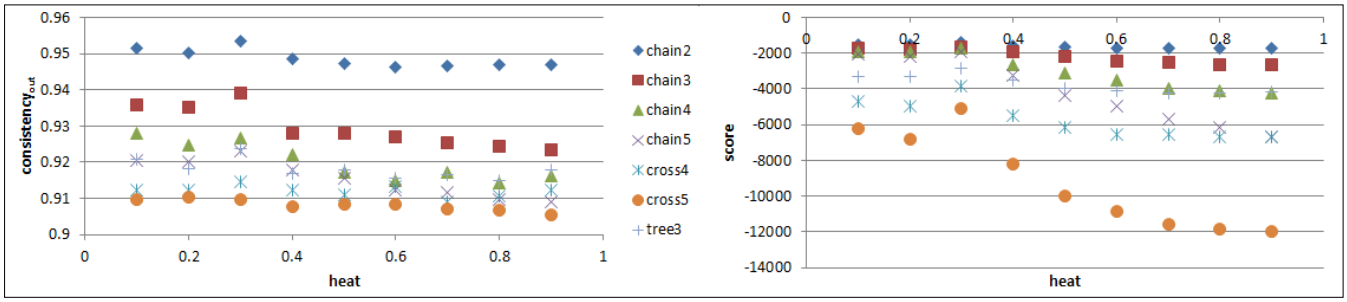

Figure 3: *Results for Test 4. Each point is an average over 25 trials.*

[2] Note that the file experiments/tests.xlsx in the submitted archive contains all the data that produced these charts, and original raw (and somewhat unreadable) data can be found in experiments/data/