Supplementary information on the Grow-Shrink strategy for learning Markov network structures constrained by context-specific independences

Alejandro Edera, Yanela Strappa, and Facundo Bromberg

Departamento de Sistemas de Información, Universidad Tecnológica Nacional, Rodriguez 273, M5502 Mendoza, Argentina
{aedera,ystrappa,fbromberg}@frm.utn.edu.ar

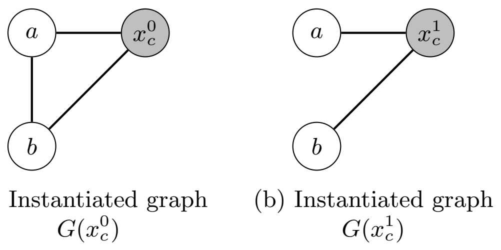
This supplement contains several important details that we cannot place in the paper due to space restrictions. These details are three-fold. First, we present an example that may help to understand the steps performed by CSGS algorithm for learning a canonical graph. This example illustrates how spurious edges are added in the grow phase and removed in the shrink phase. Second, we describe the configuration setting used for performing density estimation algorithms in order to guarantee reproducible results in our experiments. Third, we present additional empirical results that show qualitative aspects of the structures learned by the different structure learning algorithms.

1 An example of how CSGS constructs a canonical graph

This section describes how a canonical graph is constructed by CSGS. Basically, the construction consists in adding edges to an initial empty canonical graph. Proposition 1 is used as a criterion for determining the presence of any edge. As we will see below, depending on the order in which edges are added, some edges can be spurious. Therefore, a canonical graph is constructed in two phases in order to remove possible spurious edges. First, CSGS adds edges, adding nodes to blankets using Proposition 1.2; this is called the grow phase. Second, CSGS removes spurious edges, removing nodes from blankets using Proposition 1.1; this is called the shrink phase. For determining the truth value of any independence relation in Proposition 1, we use an oracle that is capable of deciding whether a(n) (in)dependence holds in the underlying structure.

Let V be $\{a, b, c\}$ with a *lexicographical order*. Suppose we have the underlying structure shown in Figure 1 that can be represented by using two instantiated graphs: $G(x^0)$ and $G(x^1)$ shown in Figures 1a and 1b, respectively. The former graph encodes no independences. In contrast, in the latter, there is no edge between the nodes a and b when c is removed, then it encodes the context-specific independence $I(X_a, X_b \mid x_c^1)$. As discussed in Section 2.1, $I(X_a, X_b \mid x_c^1)$ implies any independence of the form $I(x_a, x_b \mid x_c^1)$, where x_a and x_b can be any value of X_a and X_b, respectively. This underlying structure can also be represented by using canonical graphs of a canonical model $\mathcal{G}$. These graphs are similar to those shown in Figure 1, except that all nodes of a canonical graph are assigned. As an illustrative example, we only focus on constructing the canonical graph $G'(x) = (V, E, x) \in \mathcal{G}$ where $x = \{x_a^0, x_b^0, x_c^1\}$. $G'(x)$ can only encode the context-specific independence $I(x_a^0, x_b^0 \mid x_c^1)$ which is implicitly encoded in the instantiated graph $G(x_c^1)$.

.

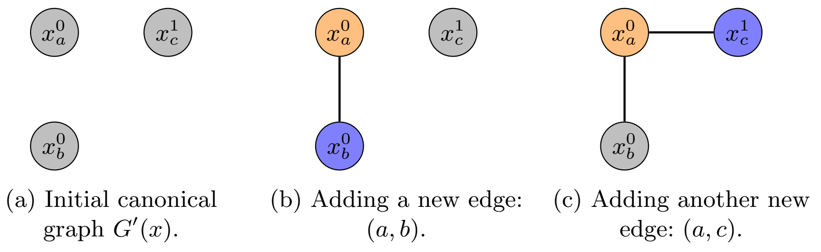


(a) Instantiated graph $G(x_c^0)$ (b) Instantiated graph $G(x_c^1)$

Fig. 1: An underlying structure represented as two instantiated graphs: $G(x_c^0)$ and $G(x_c^1)$. Gray nodes indicate assigned variables.

As is shown Figure 2a, the initial canonical graph $G'(x)$ is empty, thus $\text{MB}(a) = \{\}$ for all $a \in V$. Following the ordering, we start adding nodes to the blanket $\text{MB}(a)$ by using Proposition 1.2. We can add edges between a and any node in $V \setminus \text{MB}(a) \setminus \{a\} = \{b, c\}$. Thus, for determining the presence of the edge (a, b), we query the oracle about the assertion $I(X_a, X_b \mid x_{\text{MB}(a)})$, where $b \notin \text{MB}(a)$. The oracle answers false, thus we add the edge (a, b) to E which results in a "growth" in $\text{MB}(a) = \{b\}$. The new canonical graph is shown in Figure 2b. Next, for determining the presence of the edge (a, c), we query the oracle about the assertion $I(X_a, X_c \mid x_{\text{MB}(a)})$ where $c \notin \text{MB}(a)$. The oracle answers false, thus we add the edge (a, c) to E such as shown in Figure 2c, resulting in $\text{MB}(a) = \{b, c\}$.

.

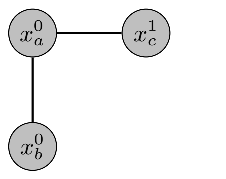


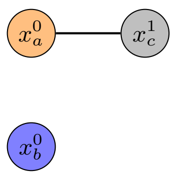
(a) Initial canonical graph $G'(x)$. (b) Adding a new edge: (a, b). (c) Adding another new edge: (a, c).

Fig. 2: Steps performed by grow phase to add nodes to the Markov blanket $\text{MB}(a)$ in the canonical graph $G'(\{x_a^0, x_b^0, x_c^1\})$. Orange and blue nodes indicate nodes involved in Proposition 1.

At the end of the grow phase, we have the canonical graph $G'(x)$ shown in Figure 3a. Comparing $G'(x)$ with $G(x_c^1)$, the edge (a,b) in $G'(x)$ is spurious because it is not in $G(x_c^1)$. Since $\mathrm{MB}(a) = \{b,c\}$, we can use Proposition 1.1 for removing the edges (a,b) and (a,c). In the first case, we query the oracle about the assertion $I(X_a, X_b \mid x_{\mathrm{MB}(a)\setminus\{b\}})$ where $b \in \mathrm{MB}(a)$. The answer is true, then we remove (a,b) from E as is shown in Figure 3b, resulting in a "shrink" in $\mathrm{MB}(a) = \{c\}$. In the second case, we query the oracle about the assertion $I(X_a, X_c \mid x_{\mathrm{MB}(a)\setminus\{c\}})$ where $c \in \mathrm{MB}(a)$. In contrast, the answer is false, then the canonical graph $G'(x)$ remains unchanged as shown in Figure 3c.

.


(a) Initial graph (output of grow phase).


(b) Removing the edge (a,b).

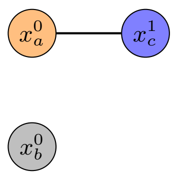

(c) The edge (a,c) cannot be removed.

Fig. 3: Steps performed by shrink phase to remove nodes from the Markov blanket $\mathrm{MB}(a)$ in the canonical graph $G'(\{x_a^0, x_b^0, x_c^1\})$.

Finally, repeating the previous steps by using the nodes b and c in turn, we obtain the canonical graph shown in Figure 4. This graph encodes the context-specific independence $I(x_a^0, x_b^0 \mid x_c^1)$.

.

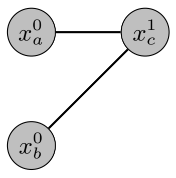

(a) The canonical graph $G'(\{x_a^0, x_b^0, x_c^1\})$.

Fig. 4: Resulting canonical graph obtained after performing grow and shrink phases.

2 Configuration setting for density estimation algorithms

We replicate the recommended tuning parameters for GSSL and DTSL algorithms detailed in [1], and [2], respectively. More precisely, for GSSL, we generated 0.5, 1, 2, and 5 million *positive features*[1]; with pruning thresholds of 1, 5, and 10; Gaussian standard deviations of 0.1, 0.5, and 1, combined with L_1 priors of 1, 5, and 10; giving a total of 108 configurations. On the other hand, for DTSL, we used the κ values of 1.0, 0.1, 0.01, 0.001, 0.0001 to obtain an initial model, and then 4 feature generation methods for positive features (NONZERO, PRUNE-NONZERO, PRUNE-5-NONZERO, PRUNE-10-NONZERO); and Gaussian standard deviations combined to L_1 priors similar to GSSL; giving a total of 50 configurations.

3 Additional empirical results

The average feature length is a useful statistic over the learned structures, which summarizes important information by determining qualitative key aspects present in the underlying structure. Therefore, these statistics allow us to evaluate if the learned structures have correctly encoded the context-specific independences present in the underlying structure. In contrast to standard metrics, such as Hamming distance and F-measure, the average feature length can be computed over complex representations of structures (such as sets of features). Thus, each learned structure is represented as a set of features $\mathcal{F}$, and then we compute the average feature length for two important subsets of their features: the features $\mathcal{F}' = \{f_D^i \in \mathcal{F} : x_w^i = x_w^0\}$ that satisfy the context x_w^0, and the features $\mathcal{F}'' = \{f_D^i \in \mathcal{F} : x_w^i = x_w^1\}$ that satisfy the context x_w^1, where $\mathcal{F}' \cap \mathcal{F}'' = \emptyset$. According to the underlying structure, these two averages are different because the context-specific independences are only present on the context x_w^1. Ideally, the average feature lengths of $\mathcal{F}'$ should be equal to n, and the average of $\mathcal{F}''$ should be equal to 2. In this manner, using both averages from the learned structures, we can determine which structure is more accurate, that is, that structure whose averages are closer to the averages obtained from the underlying structure.

Figure 5 shows the average feature lengths for all the structure learning algorithms. This figure only shows the two average feature lengths for CSGS and CSPC because, for the remaining algorithms, it is not possible to report both averages. On the one hand, for knowledge discovery algorithms, the features generated from the learned graphs result in feature lengths that are the same for both contexts. On the other hand, for density estimations algorithms, the learned features are positive, resulting in a unique feature length for $\mathcal{F}''$. As a result, in qualitative terms, the structures learned by knowledge discovery and density estimation algorithms cannot capture the context-specific independences. In contrast, CSGS and CSPC obtain different average feature lengths for each

[1] A feature is positive if its conjunction of values does not have false values; formally, a feature f_D^i is positive if $x_a^i \neq 0$ for all $a \in D$.

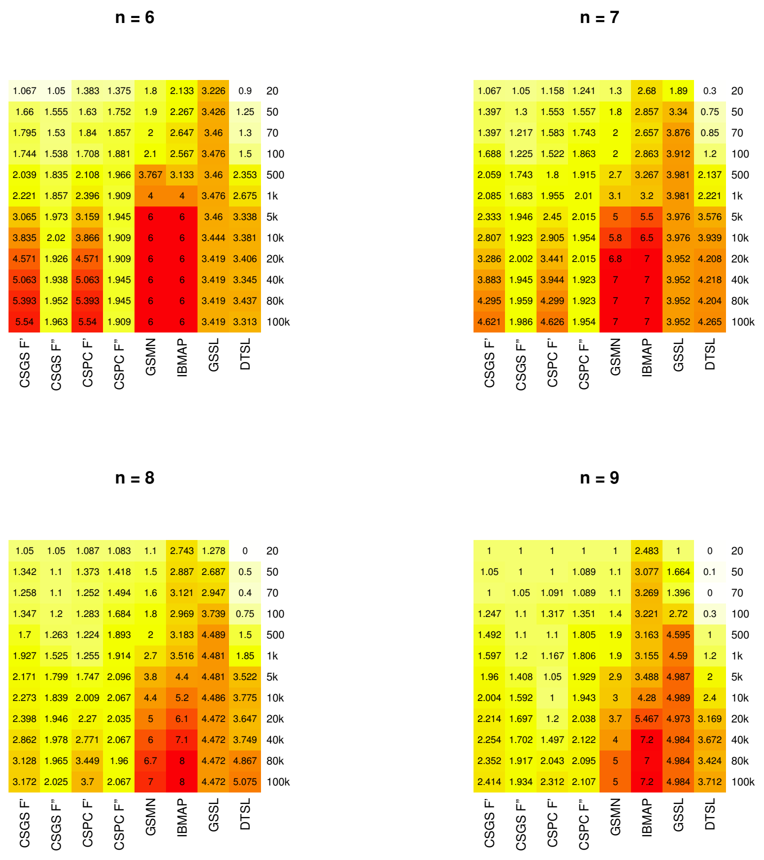
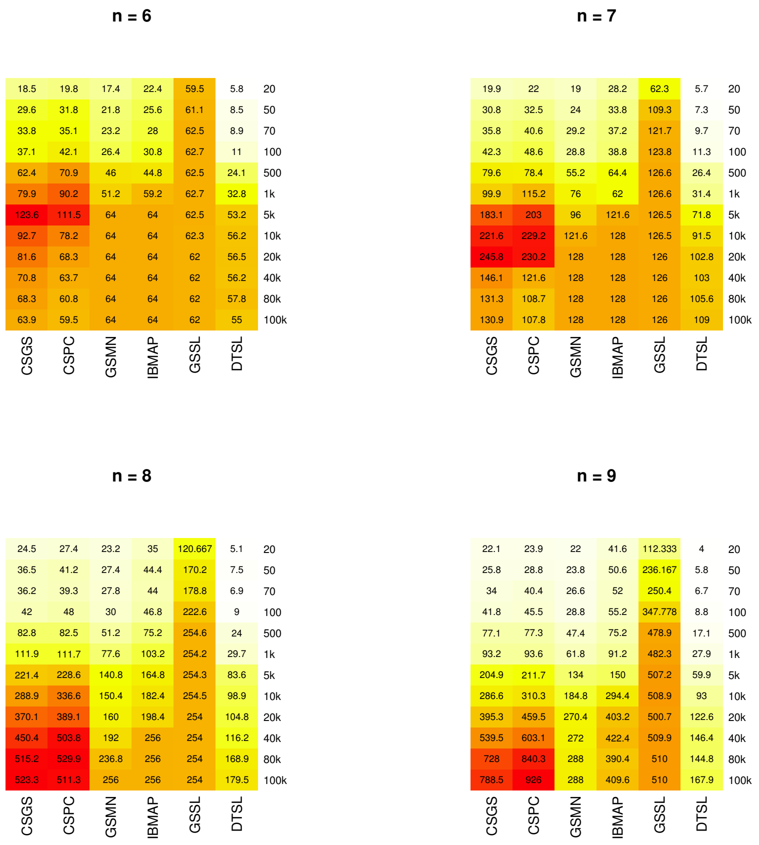


Fig. 5: Average lengths for two subsets of the features learned by the learning algorithms: the features $\mathcal{F}'$, which satisfy the context x_w^0; and the features $\mathcal{F}''$, which satisfy the context x_w^1. Only the features learned by CSGS and CSPC can satisfy both contexts. For knowledge discovery algorithms, the average feature lengths for $\mathcal{F}'$ and $\mathcal{F}''$ are the same. For density estimation algorithms, only the average feature length for $\mathcal{F}''$ can be computed. Every cell represents the average over ten datasets with a fixed size.

context. As shown in Figure 5, the averages for CSGS and CSPC follow the expected trend, that is, the average feature lengths of $\mathcal{F}'$ are greater than 2, and the averages of $\mathcal{F}''$ are close to 2. For the most reliable situations, namely $n = 6$ for $|\mathcal{D}| \geq 10k$, CSGS and CSPC obtain very accurate averages. Additionally, the average feature lengths obtained by CSGS are relatively similar to those obtained by CSPC, showing that both algorithms learn similar structures.

Another interesting trend shown in Figure 5 is in the average feature lengths obtained by knowledge discovery algorithms. Given that the underlying structure encodes no conditional independences, knowledge discovery algorithms do not learn any independence, obtaining fully connected graphs. This fact is mirrored in their average feature lengths, which tend to be close to n.



n = 6

CSGS	CSPC	GSMN	IBMAP	GSSL	DTSL	
18.5	19.8	17.4	22.4	59.5	5.8	20
29.6	31.8	21.8	25.6	61.1	8.5	50
33.8	35.1	23.2	28	62.5	8.9	70
37.1	42.1	26.4	30.8	62.7	11	100
62.4	70.9	46	44.8	62.5	24.1	500
79.9	90.2	51.2	59.2	62.7	32.8	1k
123.6	111.5	64	64	62.5	53.2	5k
92.7	78.2	64	64	62.3	56.2	10k
81.6	68.3	64	64	62	56.5	20k
70.8	63.7	64	64	62	56.2	40k
68.3	60.8	64	64	62	57.8	80k
63.9	59.5	64	64	62	55	100k

n = 7

CSGS	CSPC	GSMN	IBMAP	GSSL	DTSL	
19.9	22	19	28.2	62.3	5.7	20
30.8	32.5	24	33.8	109.3	7.3	50
35.8	40.6	29.2	37.2	121.7	9.7	70
42.3	48.6	28.8	38.8	123.8	11.3	100
79.6	78.4	55.2	64.4	126.6	26.4	500
99.9	115.2	76	62	126.6	31.4	1k
183.1	203	96	121.6	126.5	71.8	5k
221.6	229.2	121.6	128	126.5	91.5	10k
245.8	230.2	128	128	126	102.8	20k
146.1	121.6	128	128	126	103	40k
131.3	108.7	128	128	126	105.6	80k
130.9	107.8	128	128	126	109	100k

n = 8

CSGS	CSPC	GSMN	IBMAP	GSSL	DTSL	
24.5	27.4	23.2	35	120.667	5.1	20
36.5	41.2	27.4	44.4	170.2	7.5	50
36.2	39.3	27.8	44	178.8	6.9	70
42	48	30	46.8	222.6	9	100
82.8	82.5	51.2	75.2	254.6	24	500
111.9	111.7	77.6	103.2	254.2	29.7	1k
221.4	228.6	140.8	164.8	254.3	83.6	5k
288.9	336.6	150.4	182.4	254.5	98.9	10k
370.1	389.1	160	198.4	254	104.8	20k
450.4	503.8	192	256	254	116.2	40k
515.2	529.9	236.8	256	254	168.9	80k
523.3	511.3	256	256	254	179.5	100k

n = 9

CSGS	CSPC	GSMN	IBMAP	GSSL	DTSL	
22.1	23.9	22	41.6	112.333	4	20
25.8	28.8	23.8	50.6	236.167	5.8	50
34	40.4	26.6	52	250.4	6.7	70
41.8	45.5	28.8	55.2	347.778	8.8	100
77.1	77.3	47.4	75.2	478.9	17.1	500
93.2	93.6	61.8	91.2	482.3	27.9	1k
204.9	211.7	134	150	507.2	59.9	5k
286.6	310.3	184.8	294.4	508.9	93	10k
395.3	459.5	270.4	403.2	500.7	122.6	20k
539.5	603.1	272	422.4	509.9	146.4	40k
728	840.3	288	390.4	510	144.8	80k
788.5	926	288	409.6	510	167.9	100k

Fig. 6: Average number of features obtained by each learning algorithm. Every cell represents the average over ten datasets with a fixed size.

Finally, Figure 6 shows the number of features obtained by each learning algorithm. These results highlight the limitation of knowledge discovery algorithms in learning context-specific independences. Knowledge discovery algorithms only tend to learn fully connected structures, that is, a graph that encodes no independences. This fact can be analyzed by using Figures 5 and 6. For instance, in the case of GSMN for $n = 6$ and $|\mathcal{D}| \geq 5k$, the learned structure is fully connected; hence, feature lengths are equal to 6, such as shown in Figure 5, and numbers of features are $2^6 = 64$, such as shown in Figure 6.

References

1. Haaren, J.V., Davis, J.: Markov network structure learning: A randomized feature generation approach. In: Proceedings of the Twenty-Sixth National Conference on Artificial Intelligence. AAAI Press (2012)
2. Lowd, D., Davis, J.: Learning Markov network structure with decision trees. In: Data Mining (ICDM), 2010 IEEE 10th International Conference on. pp. 334–343. IEEE (2010)