Parallel Bayesian Network Structure Learning

Tian Gao* [1] **Dennis Wei*** [1]

Abstract

Recent advances in Bayesian Network (BN) structure learning have focused on local-to-global learning, where the graph structure is learned via one local subgraph at a time. As a natural progression, we investigate parallel learning of BN structures via multiple learning agents simultaneously, where each agent learns one local subgraph at a time. We find that parallel learning can reduce the number of subgraphs requiring structure learning by storing previously queried results and communicating (even partial) results among agents. More specifically, by using novel rules on query subset and superset inference, many subgraph structures can be inferred without learning. We provide a sound and complete parallel structure learning (PSL) algorithm, and demonstrate its improved efficiency over state-of-the-art single-thread learning algorithms.

1. Introduction

Bayesian networks (BN) are widely used in machine learning applications (Ott et al., 2004; Spirtes et al., 1999). The structure of a BN takes the form of a directed acyclic graph (DAG) and plays a vital part in applications such as causal inference. Many algorithms for learning BN structures from data have been developed, including score-based and constraint-based approaches (Chickering, 2002; Koivisto & Sood, 2004; Silander & Myllymaki, 2006; Jaakkola et al., 2010; Cussens, 2011; Yuan & Malone, 2013).

To alleviate the NP-hard complexity (Chickering et al., 2012) of learning a BN structure all at once, i.e. globally, both local and local-to-global learning approaches have been proposed (Niinimaki & Parviainen, 2012; Gao et al., 2017). Instead of operating on the full DAG search space over all variables, local and local-to-global methods limit the size of the space by learning local structures over smaller sets of variables. Here, local structures usually refer to the parent-child (PC) set or the Markov Blanket (MB) set (Pearl, 1988) of one or more nodes in a DAG. Local learning algorithms (Koller & Sahami, 1996; Tsamardinos et al., 2003; Fu & Desmarais, 2008) focus on a specific target variable and iteratively query other variables to learn the local structure around the target, either its PC set, MB set, or both. Many local learning algorithms have shown promising performance in practice (Aliferis et al., 2010). Using learned local structures, some prior works have taken the next step of combining them into a global structure. Several researchers (Margaritis & Thrun, 1999; Pellet & Ellisseeff, 2008) have proposed algorithms that first identify the MBs of some nodes in the graph and then connect the MBs in a maximally consistent way to learn the global structure of the BN. This approach has the benefit of improving the efficiency of exact structure learning for graphs with favorable neighbor structures (Gao et al., 2017), as at each step only a small number of variables is expected to be used.

In this paper, we consider the problem of learning a BN structure using multiple local learning agents at the same time. Each agent follows a local learning approach as described above. A naive solution would be to learn the local structure of each node using a separate agent, and then combine these local structures after learning. Given the parallel paradigm however, a more sophisticated approach would allow communication among agents both during and after each round of local structure learning. We address the question of how to maximize computational savings using such communication. In particular, we find that by tracking the query sets generated during local learning as well as the non-edges that are learned, the graphs that would result from structure learning on some future query sets can instead be more efficiently inferred from this history. We introduce two such inference rules: query subset and superset inference, and propose a sound and complete parallel BN structure learning (PSL) algorithm. The number of structure learning function calls saved by subset and superset inference is analyzed. We demonstrate that the proposed algorithm could lead to very significant savings compared to single-thread baseline methods such as graph growing SL algorithm (GGSL) (Gao et al., 2017).

Our approach is fundamentally different from and complementary to existing parallel algorithms for *global* BN

*Equal contribution [1]IBM Research, Yorktown Heights, NY 10598 USA. Correspondence to: Tian Gao <tgao@us.ibm.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

structure learning. These algorithms (Tamada et al., 2011; Nikolova et al., 2013; Misra et al., 2014) divide the DAG search space and associated computational load into multiple parts and assign agents to different parts. In contrast, our parallel algorithm does not consider the global search space, instead assigning agents to different target nodes and searching the much smaller space of local structures around each target, which are later combined. The two approaches can be combined by having each agent in our algorithm use the existing parallel algorithms to learn BN structures over the query sets that it encounters, resulting in even more parallel and efficient computation. Our approach is also different from existing parallel algorithms for other graphical models (Guestrin et al., 2003; Gonzalez et al., 2009; 2011), which are designed for inference and utilize smart parallelism in computation but do not consider directed acyclic graphs (DAG) directly.

Notation: We use capital letters (such as X, Y) to represent variables, small letters (such as x, y) to represent values of variables, and bold letters (such as $\mathbf{V}, \mathbf{MB}$) to represent variable sets. $|\mathbf{V}|$ represents the size of a set $\mathbf{V}$. $X \perp\!\!\!\perp Y$ and $X \not\perp\!\!\!\perp Y$ represent independence and dependence between X and Y, respectively.

2. Technical Preliminaries

2.1. Bayesian Networks and Structure Learning

Let $\mathbf{V}$ denote a set of random variables. A Bayesian network (BN) for $\mathbf{V}$ is represented by a pair (G, θ), where θ is a set of parameters for the associated probability distribution. Our focus is on the network structure G, a DAG with nodes corresponding to the random variables in $\mathbf{V}$. If a directed edge exists from node X to node Y in G, X is a *parent* of Y and Y is a *child* of X. More generally, X is an *ancestor* of Y and Y is a *descendant* of X if there is a sequence of edges in the same direction beginning with X and leading to Y. If X and Y have a common child and they are not adjacent, X and Y are *spouses* of each other.

The *Local Markov Condition* (Pearl, 1988) states that a node in a BN is independent of its non-descendant nodes, given its parents. A *Markov Blanket* of a variable T, $\mathbf{MB}_T$, is the minimal set of nodes conditioned on which all other nodes are independent of T, denoted as $X \perp\!\!\!\perp T | \mathbf{MB}_T, \forall X \in \{\mathbf{V} \setminus T \setminus \mathbf{MB}_T\}$. The Markov Blanket consists of the parents, children, and spouses of T.

A DAG G and a joint distribution $\mathcal{P}$ are *faithful* to each other if all and only the conditional independencies true in $\mathcal{P}$ are entailed by G (Pearl, 1988). The faithfulness condition enables us to recover G from $\mathcal{P}$. Given independent and identically distributed (i.i.d.) samples D from an unknown distribution $\mathcal{P}$, corresponding to a faithful but unknown DAG G^0, *structure learning* refers to recovering the DAG G^0 from D while *local structure learning* refers to finding the PC or MB set of a target node in G^0. To avoid symbol confusion, we use G to represent any learned DAG and use G^0 to represent the ground truth DAG.

Algorithm 1 `LocalLearn`

Input: dataset D, target node T
{step 1: find the PC set }
$\mathbf{PC}_T \leftarrow \emptyset$, $\mathbf{O} \leftarrow \mathbf{V} \setminus \{T\}$;
while $\mathbf{O}$ is nonempty **do**
  choose $X \in \mathbf{O}$, $\mathbf{O} \leftarrow \mathbf{O} \setminus \{X\}$;
  $\mathbf{Z} \leftarrow \{T, X\} \cup \mathbf{PC}_T$;
  $G \leftarrow$ `BNStructLearn` $(\mathbf{Z}, D_Z)$;
  $\mathbf{PC}_T, \mathbf{P}_T, \mathbf{C}_T \leftarrow$ `findPC`(G, T) ;
end while
{step 2: remove false PC nodes and find spouses}
$\mathbf{S}_T \leftarrow \emptyset$, $\mathbf{O} \leftarrow \mathbf{V} \setminus \mathbf{PC}_T$;
while $\mathbf{O}$ is nonempty **do**
  choose $X \in \mathbf{O}$, $\mathbf{O} \leftarrow \mathbf{O} \setminus \{X\}$;
  $\mathbf{Z} \leftarrow \{T, X\} \cup \mathbf{PC}_T \cup \mathbf{S}_T$;
  $G \leftarrow$ `BNStructLearn` $(\mathbf{Z}, D_Z)$;
  $\mathbf{PC}_T, \mathbf{P}_T, \mathbf{C}_T \leftarrow$ `findPC`(G, T) ;
  $\mathbf{S}_T \leftarrow$ `findSpouse`(G, T) ;
end while
Return: $\mathbf{MB} \leftarrow \mathbf{P}_T \cup \mathbf{C}_T \cup \mathbf{S}_T$;

Score-based structure learning algorithms use a score function $s(G, D)$ that measures the goodness of fit of a DAG G to data D, seeking a G that maximizes the score. Commonly used Bayesian score criteria, such as BDeu, are decomposable, consistent, locally consistent (Chickering, 2002), and score equivalent (Heckerman et al., 1995).

We assume the Markov condition, faithfulness condition, and infinite data size hold in the theoretical development of the paper. By using score consistency, it is easy to show that true positive edges in the ground truth DAG G^0 will always be present in a learned DAG or sub-DAG, and non-edges in learned graphs will be true negatives.

Lemma 1. ***Learned v.s. Ground Truth Edges***. *Let G^0 be a faithful DAG for distribution $\mathcal{P}$ over $\mathbf{V}$, and $G_{\mathbf{Z}}$ be the DAG learned by an exact BN structure learning algorithm over the subset of variables $\mathbf{Z} \subseteq \mathbf{V}$. Under the faithfulness and infinite data assumption, then 1) edges in subgraph $G^0_{\mathbf{Z}}$ are a subset of edges in $G_{\mathbf{Z}}$, where $G^0_{\mathbf{Z}}$ is inferred from G^0. 2) edges absent in $G_{\mathbf{Z}}$ are also absent in G^0_Z.*

2.2. Local Structure Learning

We now review a state-of-the-art algorithm, `LocalLearn`, for score-based local structure learning (Gao & Ji, 2017). A variation of `LocalLearn` is used by each learning agent in our proposed parallel algorithm. `LocalLearn` is also used in existing local-to-global algorithms (Gao et al., 2017).

In Algorithm 1, subroutine `BNStructLearn` learns an optimal DAG over a set of variables in the data, and can use any exact global BN structure learning algorithm. Different `BNStructLearn` have different performance and we analyze the algorithms independent of this subroutine. Subroutines `findPC` and `findSpouse` extract a variable T's PC set and spouse set given the adjacency matrix of a graph G. `LocalLearn` first sequentially learns the PC set by repeatedly using `BNStructLearn` on a set of nodes $\mathbf{Z}$ containing the target node T, its current PC set $\mathbf{PC}_T$, and one new query variable X. Then it uses a similar procedure to learn the spouse set and update the PC set. Although the procedure is very similar to some constraint-based structure learning methods, local score-based algorithms have been shown to enjoy soundness and completeness following only this specific procedure (Gao et al., 2017).

3. Multiple Learning Agents: Parallel BN Structure Learning

We consider the problem of learning a BN structure using $K > 1$ local learners or agents working in parallel. We focus on the setting where each agent has access to sufficient data samples for potentially all the variables. Letting $D^{N\times M}$ denote the data consisting of N sample instances of M variables, each agent k has access to $D_k^{n_k\times M} \subseteq D^{N\times M}$, where $n_k \leq N$. In developing and analyzing our algorithm in this section, we make the simplifying assumption of infinite data, i.e. $n_k \to \infty$, common to existing works.

Our starting baseline is for each agent to use `LocalLearn` to learn a local structure separately, based on the fact that it is a sound and complete algorithm for this task (Theorem 1, (Gao et al., 2017)). The local structures could then be combined into a global graph using local-to-global approaches such as SLL (Niinimaki & Parviainen, 2012) or GGSL (Gao et al., 2017). The goal in the remainder of the paper is to improve upon this baseline in terms of computational efficiency, for example by reducing the total time or number of function calls. This goal entails several research questions: how agents should be assigned to different parts of the network, how agents should communicate, and how learned structures from different agents can be merged. These questions are addressed in Section 3.2.

As discussed in Section 2.2, `LocalLearn` proceeds by performing structure learning queries on a sequence of subsets of the variables, referred to as *query sets*. We assume non-delayed communication between learning agents, meaning that agents may share information after each query rather than waiting until the completion of `LocalLearn`. We also assume that the cost of communication is minimal compared to structure learning. Thus given multiple agents with the ability to communicate while executing `LocalLearn` in parallel, the question arises as to how they may help each other in the process, and specifically how the query sets that they generate relate to each other. This question is the subject of Section 3.1.

In addition, we distinguish between synchronous and asynchronous learning. In synchronous learning, every agent performs one query and waits for other agents to finish before communicating and proceeding with the next query. Asynchronous learning means that agents do not wait for other agents to finish their queries, continuing with the next if they finish ahead of others.

3.1. Query Set Relationships

Query sets from different agents can either overlap at least partially or not overlap at all. Within the partially overlapping case, we consider the subcases in which one query set is either a subset or a superset of another. For the non-overlapping case, we conjecture that agents cannot help each other as there is no common information.

3.1.1. Query Subset Inference

Suppose that a set to be queried is a subset of an already queried set. The simplest case is when the two query sets are in fact identical. In this case, it suffices to cache the query set and learned graph each time a `BNStructLearn` query is completed, and to retrieve the graph whenever the same query set is encountered again. Such query set caching could be implemented for example using a hash table.

If the set to be queried is a proper subset of an existing query set, then the following lemma ensures that structure learning can still be skipped by inferring the result from the graph for the existing query set.

Lemma 2. *If $\mathbf{S}_i \subseteq \mathbf{S}_j$ and $\mathbf{S}_j$ has been queried before with learned graph $G_{\mathbf{S}_j}$, let $G'_{\mathbf{S}_i}$ be a subgraph for $\mathbf{S}_i$ inferred from $G_{\mathbf{S}_j}$. Then compared to $G_{\mathbf{S}_i}$, the graph that would be learned from $\mathbf{S}_i$, $G'_{\mathbf{S}_i}$ continues to contain all the true edges in $G^0_{\mathbf{S}_i}$, the ground truth subgraph, and contains no more false edges than $G_{\mathbf{S}_i}$.*

Proof. By Markov condition and score local consistency, all the dependent relationships can be identified by d-separation, given a graph $G_{\mathbf{S}_j}$ in the superset $\mathbf{S}_j$. Then all the dependence relationships among the newly queried subsets can be directly read from $G_{\mathbf{S}_j}$. Moreover, by assuming faithfulness, $G_{\mathbf{S}_j}$ captures all the dependence relationships over $\mathbf{S}_j$ and hence $G'_{\mathbf{S}_i}$ captures edges between X and Y if $X \not\perp\!\!\!\perp Y, \forall X, Y \in \mathbf{S}_i$. Moreover, in comparison to $G'_{\mathbf{S}_i}$, $G_{\mathbf{S}_i}$ can capture additional spurious relationships over $\mathbf{S}_i$ due to the removal of variables in $\mathbf{S}_j \setminus \mathbf{S}_i$. □

Lemma 2 essentially states that the graph $G'_{\mathbf{S}_i}$ inferred from $G_{\mathbf{S}_j}$, the graph for the existing query set, is no worse than

the graph $G_{\mathbf{S}_i}$ that would have been learned from the subset because of the possible inclusion of some latent variables. Moreover, $G'_{\mathbf{S}_i}$ can be efficiently inferred using d-separation from $G_{\mathbf{S}_j}$, which is much faster than the exponential complexity of local structure learning. In fact, Lemma 2 suggests maintaining a single graph G of the same size as the ground truth DAG G^0 to capture all query set results. PC and spouse relationships can then be efficiently inferred from G without necessarily inferring the full subgraph for a subset.

3.1.2. Query Superset Inference

In the case where the set to be queried is a superset of an existing query set, it may also be possible to avoid a query to `BNStructLearn` and instead infer the result from the existing graph, based on the following lemma.

Lemma 3. *Let $\mathbf{S}_i$ be an already queried set, and let node X be a node to be queried next, forming the next query set $\mathbf{S}_j \leftarrow \mathbf{S}_i \cup \{X\}$. If X is not adjacent to any node in $\mathbf{S}_i$ according to previous learning, then the graph over $\mathbf{S}_j$, $G_{\mathbf{S}_j}$, can be inferred from $G_{\mathbf{S}_i}$ over $\mathbf{S}_i$.*

Proof. By Lemma 1, true positive edges will always be reflected in any learned subgraph of $G_{\mathbf{S}_j}$. Since X and S_i are not adjacent according to previous learning, they cannot be adjacent in $G_{\mathbf{S}_j}$ or any supergraph of $G_{\mathbf{S}_j}$. □

Hence based on Lemma 3, even if $X \cup \mathbf{S}_i$ is never queried jointly, from existing learned subgraphs it is possible to identify the independence relationships between X and other variables. If X is not adjacent to any variable in $\mathbf{S}_i$, the independence relationships have to hold even if $\mathbf{S}_j$ is queried jointly. However, if X is adjacent to at least one variable in $\mathbf{S}_i$, a new query would be needed as the exact location of X in the graph over set $\mathbf{S}_j$ would be unclear.

3.2. Proposed Algorithm

Based on the preceding discussion, we propose a novel parallel structure learning algorithm (PSL), as shown in Algorithm 2. Below we explain the different elements of PSL.

Shared Memory. The learning agents share information with each other through two objects, a graph adjacency matrix G, and query set history $\mathcal{H}$. The adjacency matrix G is continuously updated by the agents to represent the current knowledge of the graph, per discussion in Section 3.1. Each element $G(i,j)$, $i,j \in V$, can take 4 different values with the following meanings: $G(i,j) = G(j,i) = 0$ indicates that there is definitely no edge between nodes i and j. $G(i,j) = 1$ and $G(j,i) = 0$ means that there is definitely a directed edge from i to j, while $G(i,j) = G(j,i) = 1$ means that there is a definite edge between i and j but the direction is unknown. $G(i,j) = 2$ indicates a possible but not confirmed edge, an intermediate product of the learning procedure. Lastly, $G(i,j)$ is initialized to -1 to represent unknown and non-queried edges. The query history $\mathcal{H}$ is a collection of past query sets, i.e. subsets of nodes for which the BN structure has been learned via `BNStructLearn`. $\mathcal{H}$ is also continuously updated by the agents.

Algorithm 2 `Parallel BN Structure Learning`

Input: dataset D for all variables $\mathbf{V}$, K learning agents, exploitation probability p
$G(i,j) \leftarrow -1 \; \forall \, i,j \in \mathbf{V}$
$\mathbf{A} \leftarrow \emptyset$
$\mathbf{H} \leftarrow \emptyset$
while $|\mathbf{A}| < |\mathbf{V}| - 1$ **and** agent is available **do**
  $T \leftarrow$ `chooseTarget`$(G, \mathbf{A}, p)$
  $\mathbf{MB}_T \leftarrow$ `findMB`(G, T)
  $\mathbf{O} \leftarrow \mathbf{MB}_T \cup (\mathbf{V} \backslash \{T, \mathbf{A}\})$
  $\mathbf{O} \leftarrow$ `orderQueries`$(\mathbf{O}, T, G, \mathbf{A})$
  Assign to agent:
  $G, \mathbf{H} \leftarrow$ `ParallelLocalLearn`$(D_Z, T, \mathbf{O}, G, \mathbf{H})$
  $\mathbf{A} \leftarrow \mathbf{A} \cup \{T\}$
end while
$G \leftarrow$ `PDAG-to-DAG`(G)
Return: G

Agent Coordination. Algorithm 2 includes several coordinating tasks, which could be performed by a central agent. These are 1) assigning available agents to new target nodes T, 2) tracking the set of previous targets $\mathbf{A}$, 3) forming the set of non-target variables $\mathbf{O}$ that an agent will use in local structure learning, and 4) determining the order in which an agent will query variables in $\mathbf{O}$. Task 3) is done in the same way as in (Gao et al., 2017); namely, previous targets that are not in the MB of T (according to the current G) are excluded for efficiency. The following paragraphs describe tasks 1) and 4) in more detail. The algorithm continues until all but one node has been selected as target.

Target Selection. The selection of targets for agents (function `chooseTarget` in Algorithm 2) can be seen as a trade-off between exploitation and exploration. Exploitation in this context refers to choosing targets that are adjacent to previous targets on the expectation that their local structures can be more easily learned by leveraging existing knowledge. Exploration on the other hand means choosing targets whose local structure is relatively unknown or that are "far away" in the sense of being relatively disconnected from previous targets. In both cases, exploring such targets is expected to yield more new information. In the multi-agent setting, this motivates assigning agents to different parts of the graph so that the information they exchange is more diverse and could lead to more savings in later iterations.

We propose two strategies for selecting targets that make the above intuition concrete. Further justification for these

strategies is provided in Section 3.3. In the first strategy corresponding to exploitation, a new target is chosen from the largest of the MB sets of previous targets,

$$T \in \mathbf{MB}\left(\arg\max_{A\in\mathbf{A}} |\mathbf{MB}_A|\right). \quad (1)$$

The specific member of the MB can be chosen arbitrarily, as can be the largest MB set when it is not unique. In the second strategy corresponding to exploration, a new target is chosen as the node with the most -1 and 0 elements in the current adjacency matrix,

$$T \in \arg\max_{i\in\mathbf{V}} |\{j : G_{ij} \leq 0\}|. \quad (2)$$

In other words, it is the node with the most non-queried edges plus known non-adjacencies. Again ties can be broken arbitrarily.

A more sophisticated version of (1) accounts for the fact that Algorithm 2 excludes from the set $\mathbf{O}$ previous targets that are not in $\mathbf{MB}_T$, i.e. $\mathbf{A}\backslash\mathbf{MB}_T$. Thus (1) becomes a joint maximization over previous MBs and members thereof,

$$T, A^* \in \arg\max \{|\mathbf{MB}_A\backslash(\mathbf{A}\backslash\mathbf{MB}_{T'})| : T' \in \mathbf{MB}_A, A \in \mathbf{A}\}. \quad (3)$$

In our implementation, both the simulations in Section 4 and analysis in Section 3.3 use the more refined version (3).

To trade off exploitation for exploration, we propose a simple randomization: With probability p, the function `chooseTarget` in Algorithm 2 returns a target chosen according to (1) or (3), and with probability $1-p$, it returns the result of (2).

Initial Target Selection. At the beginning of Algorithm 2 when the adjacency matrix G is completely unknown and all agents are available, the preceding target selection strategies do not favor any one node. Hence the initial selection chooses K of the nodes uniformly at random as targets.

Query Order. To complement the target selection strategies in (2)–(3), the function `orderQueries` orders non-target variables in $\mathbf{O}$, which consists of the currently learned MB_T of T and other non-queried variables, in a particular way. `findMB` is an extraction function that reads of MB_T from the current learned G. Section 3.3 explains how the target selection and query order work together. `orderQueries` sorts the variables in $\mathbf{O}$ into three groups. Variables within a group can be in any order. The first group consists of nodes known to not be adjacent to T, i.e. nodes i for which $G_{Ti} = 0$ in the current adjacency matrix. Next, if T was chosen using (1) or (3) and $A^* \in \mathbf{A}$ denotes the previous target with the maximizing MB set, then the second group is set to be

$$(\{A^*\} \cup \mathbf{MB}_{A^*})\backslash(\{T\} \cup \mathbf{A}\backslash\mathbf{MB}_T). \quad (4)$$

The above differs slightly from the expression in (3) in that T is now excluded while A^* is included. Otherwise if (2) was used, the second group is chosen as the largest of the previous MBs that contain T, again accounting for the removal of T and $\mathbf{A}\backslash\mathbf{MB}_T$ as in (4), i.e. the nodes corresponding to

$$\max\{|(\{A\} \cup \mathbf{MB}_A)\backslash(\{T\} \cup \mathbf{A}\backslash\mathbf{MB}_T)| : A \in \mathbf{A}, T \in \mathbf{MB}_A\}.$$

The remaining variables in $\mathbf{O}$ form the third group.

Local Structure Learning. The subroutine `ParallelLocalLearn` (Algorithm 3) is an adaptation of `LocalLearn` from (Gao et al., 2017) for learning the local structure of a target node, taking into account the aid of other agents and existing knowledge. Like `LocalLearn`, `ParallelLocalLearn` consists of two steps, one loop to learn the PCs of the target, and a second loop to learn the spouses and remove false PCs. The differences from `LocalLearn` are the ordering of the query variables in $\mathbf{O}$, already discussed, and the application of query subset and superset inference from Section 3.1. Specifically, each time a query set $\mathbf{Z}$ is formed in both the PC and spouse steps, the algorithm checks whether the new node i is not adjacent to any other nodes in $\mathbf{Z}$ and whether $\mathbf{Z}$ is a subset of a previous query set $\mathbf{Z}_H$ in the history $\mathcal{H}$. If the first case is true, then by Lemma 3, the query set can be skipped. If the second case is true, then by Lemma 2, the sets $\mathbf{PC}_T$ and $\mathbf{S}_T$ can be updated by reading from the subgraph corresponding to $\mathbf{Z}_H \supseteq \mathbf{Z}$.

If neither subset nor superset inference apply, then the query set $\mathbf{Z}$ is passed to `BNStructLearn` as before to learn its structure. Upon completion, the sets $\mathbf{PC}_T$ and $\mathbf{S}_T$ and the global adjacency matrix G are updated, and $\mathbf{Z}$ is added to the history $\mathcal{H}$.

Graph Updating. The function `updateGraph` can change the values of $G(i, j)$ according to the following rules: -1 can be changed to 0 or 2 but not to 1 directly. The reason is that confirmation of an edge requires at least a second pass in the form of Step 2 of Algorithm 3, and a change from -1 is by definition a first pass. 2 can be changed to 0 (false positive) but 0 can never be changed (no false negatives). Lastly, 2 can be changed to 1 only when `ParallelLocalLearn` is complete (last `updateGraph` in Algorithm 3) and only for edges incident to the target. These rules can be justified by properties of `(Parallel)LocalLearn` established in (Gao et al., 2017), particularly Lemma 2 and Theorem 1 therein. `updateGraph` can also reorient edges in the same manner as discussed in (Gao et al., 2017). In the end of Algorithm 2, running Meek-rules (Meek, 1995) would be similarly necessary.

Since the PSL algorithm follows the same learning framework as GGSL, in particular for $K = 1$ agent, and the

Algorithm 3 `ParallelLocalLearn`

Input: dataset D, target node T, ordered query list $\mathbf{O}$, current global graph G, query history $\mathbf{H}$
{Step 1: Find the PC set }
$\mathbf{PC}_T \leftarrow \emptyset$;
for $i \in \mathbf{O}$ **do**
  {superset inference}
  $\mathbf{Z} \leftarrow \{T\} \cup \mathbf{PC}_T$
  if i is not adjacent to $\mathbf{Z}$ in G **then**
    continue
  end if
  $\mathbf{Z} \leftarrow \mathbf{Z} \cup \{i\}$;
  {subset inference}
  if $\exists\, \mathbf{Z}_H \in \mathbf{H}$ such that $\mathbf{Z} \subseteq \mathbf{Z}_H$ **then**
    Update $\mathbf{PC}_T$ by reading $G(\mathbf{Z}_H)$
  else
    $G_T \leftarrow$ `BNStructLearn` $(\mathbf{Z}, D_Z)$;
    $\mathbf{PC}_T \leftarrow$ `findPC`(G_T, T) ;
    $G \leftarrow$ `updateGraph`(G, G_T)
    $\mathbf{H} \leftarrow \{\mathbf{H}, \mathbf{Z}\}$
  end if
end for
{Step 2: Remove false PC nodes and find spouses}
$\mathbf{S}_T \leftarrow \emptyset$, $\mathbf{O} \leftarrow \mathbf{O} \setminus \mathbf{PC}_T$;
for $i \in \mathbf{O}$ **do**
  {superset inference}
  $\mathbf{Z} \leftarrow \{T\} \cup \mathbf{PC}_T \cup \mathbf{S}_T$
  if i is not adjacent to $\mathbf{Z}$ in G **then**
    continue
  end if
  $\mathbf{Z} \leftarrow \mathbf{Z} \cup \{i\}$;
  {subset inference}
  if $\exists\, \mathbf{Z}_H \in \mathbf{H}$ such that $\mathbf{Z} \subseteq \mathbf{Z}_H$ **then**
    Update $\mathbf{PC}_T, \mathbf{S}_T$ by reading $G(\mathbf{Z}_H)$
  else
    $G_T \leftarrow$ `BNStructLearn` $(\mathbf{Z}, D_Z)$;
    $\mathbf{PC}_T, \mathbf{S}_T \leftarrow$ `findMB`(G_T, T) ;
    $G \leftarrow$ `updateGraph`(G, G_T)
    $\mathbf{H} \leftarrow \{\mathbf{H}, \mathbf{Z}\}$
  end if
end for
$G \leftarrow$ `updateGraph`(G, G_T)
Return: $G, \mathbf{H}$

proposed subset and superset inference rules are sound by Lemma 2 and 3, it is straightforward to show PSL's soundness and completeness.

Proposition 1. ***Soundness and Completeness***. *Under the infinite data and faithfulness assumptions, PSL (Algorithm 2) learns all and only the true edges in the underlying DAG G^0, up to the Markov equivalent class of G^0.*

Differences with Local-to-Global Learning. The proposed PSL algorithm differs from the state-of-the-art local-to-global learning algorithm GGSL (Gao et al., 2017) even in the single-agent $K = 1$ case. Most notably, in GGSL the single agent does not interact with the learning history (in the form of the current DAG) during `LocalLearn`, only afterward. GGSL does not use subset or superset inference to save on structure learning queries. In addition, GGSL uses a simpler target selection strategy of choosing from neighbors of previous targets, and it does not order queries in any way.

3.3. Computational Savings

Query subset and superset inference as incorporated in `ParallelLocalLearn` result in computational savings over the state-of-the-art GGSL algorithm (Gao et al., 2017), specifically in terms of the number of calls to `BNStructLearn`. This subsection characterizes the savings attributable to the proposed target selection strategies and query order. The cases of subset inference and superset inference are discussed separately.

Additional savings occur whenever query subset or superset inference is triggered within `ParallelLocalLearn`, beyond those due to target selection and query order analyzed below. It is more difficult however to predict when these instances will occur, especially with multiple agents exchanging information in real time. We evaluate empirically the total savings over GGSL in Section 4.

First, we state some facts about `LocalLearn`, which apply also to `ParallelLocalLearn`.

Lemma 4. *: The query sets $\mathbf{Z}$ generated in the course of `LocalLearn` have the following properties:*

1. *By the end of step 1 (PC step), there is at least one query set containing all the (true) PCs of the target T.*
2. *By the end of step 2 (spouse step), there is at least one query set containing all the PCs and spouses of T.*
3. *By the end of step 2 (spouse step) and for all nodes X that are not PCs of T, there is at least one query set containing all the PCs of T plus X.*

Proof. Properties 1 and 2 are corollaries of Lemma 2 (preservation of true PCs) and Theorem 1 (soundness and completeness) from (Gao et al., 2017). For property 3, if X is a false PC at the end of step 1, then the last query set in step 1 contains X and all true PCs from property 1. Otherwise, step 2 iterates through all remaining X and does not drop any true PCs of T. □

3.3.1. Subset Inference

The following result pertains equally to the single-agent and multiple-agent cases.

Proposition 2. *Assume that a new target T is chosen according to (3) and that its query order is determined by the function* `orderQueries`*. Then learning the local structure of T using* `ParallelLocalLearn` *saves at least*

$$\max\{|\mathbf{MB}_A \backslash (\mathbf{A} \backslash \mathbf{MB}_{T'})| : T' \in \mathbf{MB}_A, A \in \mathbf{A}\}$$

calls to `BNStructLearn` *compared to* `LocalLearn`*.*

Proof. Denote by A^* the previous target that maximizes (3) jointly with T. By Lemma 4.2, there exists a previous query set $\mathbf{Z}_H$ containing A^* and $\mathbf{MB}_{A^*} = \mathbf{PC}_{A^*} \cup \mathbf{S}_{A^*}$, which also includes $T \in \mathbf{MB}_{A^*}$ according to (3). By definition of the function `orderQueries`, the set defined in (4) is also placed early in the query order for target T, after nodes known not to be adjacent to T (which do not contribute to $\mathbf{PC}_T$) and before all other nodes. Hence during step 1 (PC step) of Algorithm 3, as long as the query variable i belongs to the set in (4), all query sets $\mathbf{Z}$ must be subsets of $\mathbf{Z}_H$ since the nodes can be drawn from at most $\{A^*\} \cup \mathbf{MB}_{A^*} \subseteq \mathbf{Z}_H$. Subset inference therefore avoids a number of calls to `BNStructLearn` at least equal to the cardinality of the set in (4). This cardinality expression can be simplified slightly by noting that the inclusion of A^* always increases the cardinality by 1, which is offset by the exclusion of T. □

Proposition 2 supports the intuition that the savings of subset inference are greater when a large MB is discovered early in Algorithm 2 (so that $\mathbf{A}$ is also small). A similar but weaker result holds if a target is chosen according to (1).

For $K > 1$, there is a trade-off between having multiple agents working early in the algorithm, when there is little knowledge to share, versus increased savings due to subset inference later in the algorithm. These later benefits are as follows: First, more query history becomes available since sets are queried at a faster rate. In particular, the early discovery of a large MB becomes more likely, especially if agents are exploring different parts of the graph. Furthermore, Proposition 2 relies only on results available at the end of a run of `ParallelLocalLearn`, specifically Lemma 4. But since agents also update the graph G and query history $\mathcal{H}$ throughout `ParallelLocalLearn`, it becomes possible to exploit subset inference immediately as knowledge becomes available, thus yielding additional savings.

3.3.2. Superset Inference

The following result applies to all target selection strategies and to the single-agent and multi-agent settings.

Proposition 3. *Assume that the query order for a target T is determined by the function* `orderQueries`*. Then query superset inference within* `ParallelLocalLearn` *saves at least $|\{i \in \mathbf{O} : G_{Ti} = 0\}|$ calls to* `BNStructLearn` *compared to* `LocalLearn`*, where G is the current graph adjacency matrix.*

Proof. Let T play the role of X in Lemma 3 and $\mathbf{S} = \{i \in \mathbf{O} : G_{Ti} = 0\}$, i.e., nodes currently known to be non-adjacent to T. `orderQueries` places the set $\mathbf{S}$ first in the query order. Hence during step 1 (PC step) of Algorithm 3, as long as the query variable $i \in \mathbf{S}$, Lemma 3 can be repeatedly invoked to avoid calls to `BNStructLearn`. □

According to Proposition 3, calls to `BNStructLearn` can be avoided whenever nodes are known to not be adjacent to a target. In the most conservative case, such nodes are guaranteed to be identified if `ParallelLocalLearn` is run with each such node as the target. This is because `ParallelLocalLearn` finds all and only the MB nodes of its target, and by extension, all non-adjacent nodes as well. It is not necessary however to wait for multiple runs of `ParallelLocalLearn` to benefit from superset inference; in fact, it is possible to do so after a single run. According to Lemma 4.3, by the end of `ParallelLocalLearn` and for each non-PC X of the target T, there is at least one query set containing X and all the PCs of T. From the results of `BNStructLearn` on these query sets, it is possible that some X are found to be non-adjacent to one or more PCs of T in addition to T itself. Choosing as new target the X that is non-adjacent to the most PCs of T may then lead to significant savings. It is difficult however to guarantee that any nodes non-adjacent to PCs of T will be found because, unlike for T itself, `ParallelLocalLearn` does not guarantee anything regarding the learned local structure around PCs of the target.

With multiple agents, there is the same trade-off as with subset inference between more work done early in the algorithm versus greater savings later. In particular, more agents means that `ParallelLocalLearn` definitively learns the non-adjacencies of more target nodes, which enables more superset inference later. The rate at which.non-adjacencies in general are learned also increases, i.e. those that are more a by-product of `ParallelLocalLearn` as discussed above.

Note that keeping the history $\mathcal{H}$ of queried subsets requires additional memory. The memory size can become big for large networks. But since it is guided by the query set learning, if the variable size becomes big, the memory requirement for `BNStructLearn` would be exponential anyway. Hence we believe `BNStrucLearn` subroutines would be the dominating memory limiting factor in large BNs. In addition, If needed, the algorithm could use multiple machines (even with limited memory) as different learning agents instead of using one single machine, which is natural for the parallel algorithm.

4. Experiments

We test the proposed PSL algorithm with different numbers K of agents and compare to the baseline GGSL (Gao et al., 2017) algorithm on the benchmark ALARM dataset. For additional results on other datasets, please refer to supplementary material. Both algorithms do not utilize seqsets used in GGSL. We use an existing Dynamic Programming algorithm (Silander & Myllymaki, 2006) as the `BNStructLearn` subroutine for all experiments. Similar performance should be obtained with other structure learning subroutines. We use 1000 samples from the dataset and perform structure learning with different algorithms. We use the BDeu score of the learned DAG over the 1000 samples as a metric for accuracy, and show the differences in BDeu scores with respect to GGSL. In addition, we also show the total number of function calls to `BNStructLearn` and the mean number of calls for each agent, as proxies for "total agent time" and the actual "running time" (time to completion) respectively. The differences between GGSL and PSL for $K = 1$ agent show the pure efficiency gain due to query subset and superset inference during the learning process. For $K > 1$ agents, the parallelism gain can be seen in the mean number of function calls. The experiments are conducted on a machine with a 2.3GHz Intel i5-5300U processor.

Table 1. BDeu Scores and Numbers of Queries for Different Parallel Algorithms on ALARM dataset.

k AGENT	SYNCHRONOUS BDEU CHANGE	MEANFC	TOTALFC
GGSL	+0.0	1418	1418
1	+0.0	369	369
2	+305.3	205.5	411
3	+ 51.5	154.0	462
4	+7.4	112.0	448
5	+225.7	100.6	503
k AGENT	ASYNCHRONOUS BDEU CHANGE	MEANFC	TOTALFC
2	+95.9	258.5	517
3	+12.4	142.3	427
4	+ 2.2	122.3	489
5	+193.0	106.2	531

For the first set of experiments, we evaluate the performance of the algorithms with respect to the number of learning agents. We test using both synchronous learning, where the central agent waits for all learning agents to finish one round of queries before starting the next round, and asynchronous learning, where learning agents run continuously without waiting. We fix $p = 0.7$ for target selection function `chooseTarget`. The results are shown in Table 4. BDeu scores fluctuates but change only minorly ($\sim 1\%$ original score in GGSL). We can see from the $K = 1$ case that the use of subset and superset inference reduces the total number of function calls by around 3 to 4 times. For $K > 1$, the parallelism gain also increases significantly with K, although the speed-up is not quite $\frac{1}{K}$. The overhead is likely due to having only limited information in the early stages of PSL. For $K \geq 4$, PSL reduces the learning time by one order of magnitude compared to GGSL. If the total number of function calls is a more important metric than the learning time, it may also be possible to further reduce the former by not starting the agents all at the same time but rather in stages. For example, the second agent may start after the first finishes one run of `ParallelLocalLearn`, the third agent waits for the previous two to finish another, and so on. This may reduce overhead in the beginning but still increase savings later.

Table 2. BDeu Scores and Numbers of Queries on ALARM dataset for 2-Agent PSL Algorithms with Different p.

p	SYNCHRONOUS BDEU CHANGE	MEANFC	TOTALFC
0.1	+20.8	249.5	499
0.3	+143.5	230	460
0.5	+123.5	211.5	423
0.7	+305.3	205.5	411
0.9	-385.8	195.5	393

In addition, we also tested the effect of the exploitation probability p on the performance of the PSL algorithm with $K = 2$ agents. As shown in Table 2, for this particular ALARM dataset, the higher the probability of exploitation, the more gain in efficiency. We suspect the reason is that ALARM could be a relatively densely connected graph. For different graph sparsities or clique sizes, the performance may change accordingly.

5. Discussion and Conclusion

We have proposed a parallel structure learning (PSL) algorithm for Bayesian networks that employs multiple agents. Novel query subset and superset inference rules were developed to reduce the number of sets for which structure learning must be performed. Target selection and query order strategies were also proposed to promote such savings. Experiments indicate that both the inference rules as well as parallelism itself contribute to significant savings in the number of structure learning queries and the time to learn the global structure.

For future work, as mentioned in Section 4, agent staging strategies could be developed to further optimize the total number of queries rather than the time to completion. This work also suggests extensions to the more difficult *distributed* setting in which agents have access to only a subset of the variables and may also be limited in communication.

Acknowledgements

We thank anonymous reviewers for many helpful comments.

References

Aliferis, C. F., Statnikov, A., Tsamardinos, I., Mani, S., and Koutsoukos, X. D. Local causal and markov blanket induction for causal discovery and feature selection for classification part i: Algorithms and empirical evaluation. *Journal of Machine Learning Research*, pp. 171–234, Jan 2010.

Chickering, D. M. Optimal structure identification with greedy search. *Journal of Machine Learning Research*, 2002.

Chickering, D. M., Meek, C., and Heckerman, D. Large-sample learning of bayesian networks is np-hard. *CoRR*, abs/1212.2468, 2012.

Cussens, J. Bayesian network learning with cutting planes. In *Proceedings of the Twenty-Seventh Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-11)*, pp. 153–160, Corvallis, Oregon, 2011. AUAI Press.

Fu, S. and Desmarais, M. C. Fast markov blanket discovery algorithm via local learning within single pass. In *Proceedings of the Canadian Society for computational studies of intelligence, 21st conference on Advances in artificial intelligence*, Canadian AI'08, Berlin, Heidelberg, 2008. Springer-Verlag.

Gao, T. and Ji, Q. Efficient score-based markov blanket discovery. *International Journal of Approximate Reasoning*, 80:277–293, 2017.

Gao, T., Fadnis, K., and Campbell, M. Local-to-global bayesian network structure learning. In *International Conference on Machine Learning*, pp. 1193–1202, 2017.

Gonzalez, J., Low, Y., and Guestrin, C. Residual splash for optimally parallelizing belief propagation. In *Artificial Intelligence and Statistics*, pp. 177–184, 2009.

Gonzalez, J., Low, Y., Gretton, A., and Guestrin, C. Parallel gibbs sampling: From colored fields to thin junction trees. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 324–332, 2011.

Guestrin, C., Koller, D., Parr, R., and Venkataraman, S. Efficient solution algorithms for factored mdps. *Journal of Artificial Intelligence Research*, 19:399–468, 2003.

Heckerman, D., Geiger, D., and Chickering, D. M. Learning bayesian networks: The combination of knowledge and statistical data. *Machine learning*, 20(3):197–243, 1995.

Jaakkola, T., Sontag, D., Globerson, A., and Meila, M. Learning bayesian network structure using lp relaxations. 2010.

Koivisto, M. and Sood, K. Exact bayesian structure discovery in bayesian networks. *The Journal of Machine Learning Research*, 5:549–573, 2004.

Koller, D. and Sahami, M. Toward optimal feature selection. In *ICML 1996*, pp. 284–292. Morgan Kaufmann, 1996.

Margaritis, D. and Thrun, S. Bayesian network induction via local neighborhoods. In *Advances in Neural Information Processing Systems 12*, pp. 505–511. MIT Press, 1999.

Meek, C. Strong completeness and faithfulness in bayesian networks. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pp. 411–418. Morgan Kaufmann Publishers Inc., 1995.

Misra, S., Vasimuddin, M., Pamnany, K., Chockalingam, S. P., Dong, Y., Xie, M., Aluru, M. R., and Aluru, S. Parallel bayesian network structure learning for genome-scale gene networks. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, pp. 461–472. IEEE Press, 2014.

Niinimaki, T. and Parviainen, P. Local structure disocvery in bayesian network. In *Proceedings of Uncertainy in Artifical Intelligence, Workshop on Causal Structure Learning*, pp. 634–643, 2012.

Nikolova, O., Zola, J., and Aluru, S. Parallel globally optimal structure learning of bayesian networks. *Journal of Parallel and Distributed Computing*, 73(8):1039–1048, 2013.

Ott, S., Imoto, S., and Miyano, S. Finding optimal models for small gene networks. In *Pacific symposium on biocomputing*, volume 9, pp. 557–567, 2004.

Pearl, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers, Inc., 2 edition, 1988.

Pellet, J.-P. and Ellisseeff, A. Using markov blankets for causal structure learning. *Journal of Machine Learning Research*, 2008.

Silander, T. and Myllymaki, P. A simple approach for finding the globally optimal bayesian network structure. In *Proceedings of the Twenty-Second Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 445–452, 2006.

Spirtes, P., Glymour, C. N., and Scheines, R. *Computation, Causation, and Discovery*. AAAI Press, 1999.

Tamada, Y., Imoto, S., and Miyano, S. Parallel algorithm for learning optimal bayesian network structure. *Journal of Machine Learning Research*, 12(Jul):2437–2459, 2011.

Tsamardinos, I., Aliferis, C., Statnikov, A., and Statnikov, E. Algorithms for large scale markov blanket discovery. In *In The 16th International FLAIRS Conference, St*, pp. 376–380. AAAI Press, 2003.

Yuan, C. and Malone, B. Learning optimal bayesian networks: A shortest path perspective. 48:23–65, 2013.