MrPC: Causal Structure Learning in Distributed Systems

Thin Nguyen[1(✉)], Duc Thanh Nguyen[2], Thuc Duy Le[3], and Svetha Venkatesh[1]

[1] Applied Artificial Intelligence Institute (A2I2), Deakin University, Melbourne, Australia
thin.nguyen@deakin.edu.au
[2] School of Information Technology, Deakin University, Melbourne, Australia
[3] School of Information Technology and Mathematical Sciences, University of South Australia, Adelaide, Australia

Abstract. **PC** algorithm (**PC**) – named after its authors, Peter and Clark – is an advanced constraint based method for learning causal structures. However, it is a time-consuming algorithm since the number of independence tests is exponential to the number of considered variables. Attempts to parallelise **PC** have been studied intensively, for example, by distributing the tests to all computing cores in a single computer. However, no effort has been made to speed up **PC** through parallelising the conditional independence tests into a cluster of computers. In this work, we propose **MrPC**, a robust and efficient **PC** algorithm, to accelerate **PC** to serve causal discovery in distributed systems. Alongside with **MrPC**, we also propose a novel manner to model non-linear causal relationships in gene regulatory data using kernel functions. We evaluate our method and its variants in the task of building gene regulatory networks. Experimental results on benchmark datasets show that the proposed **MrPC**gains up to seven times faster than sequential **PC** implementation. In addition, kernel functions outperform conventional linear causal modelling approach across different datasets.

Keywords: Causality · Explainable AI · Causal structure learning · Distributed systems

1 Introduction

PC is an important algorithm in learning causal structures for observational data [1,8]. However, the algorithm is also well-known for its high computational complexity as the number of independence tests is exponential to the number of considered variables in the worst case. This limits the applicability of the **PC** algorithm in practice. To address this issue, parallelising the algorithm is often applied, such as in [4], where the authors distributed the independence tests as tasks fed into a multi-core computer. However, there is no method speeding the **PC** algorithm in distributed systems, for example, on a cluster of computers.

H. Yang et al. (Eds.): ICONIP 2020, CCIS 1332, pp. 87–94, 2020.
https://doi.org/10.1007/978-3-030-63820-7_10

1.1 Related Work

There have been several causal discovery and causal inference methods that use **PC** as the core component and therefore they will be benefited from an efficient **PC**. Some methods have been proposed to improve the efficiency of **PC** by introducing heuristic methods, but they rather compromise the accuracy [7]. Meanwhile, others [4] aimed to parallelise the conditional independence tests into different cores of a computer, limiting the scale of the parallelisation.

Advances in cluster computing can help parallelise the conditional independence tests in **PC** algorithm into a set of connected computers. Among the most popular architectures is MapReduce [2], a distributed computing programming model, designed to enable and simplify parallel programming. The framework has been implemented and extended in several distributed systems, including Apache Spark [9], where RDD (Resilient Distributed Dataset) is proposed to keep data in-memory for faster computation.

To verify the independence/dependence of variables from coincident data, conditional independence tests are conducted. Conventionally, independence tests are performed via correlation matrices [6], which implicitly impose a linear relationship between the factors. However, this assumption is not always true in reality and unknown generally.

1.2 Contributions

In this paper, we investigate learning causal structures in distributed systems. The contributions of our work are two-fold as follows.

- First, we propose **MrPC** to perform the **PC** algorithm [1] on Apache Spark [9], a cluster computing framework supporting parallel, distributed computation and storage on multiple computers. Extending MapReduce and equipping it with in-memory computing capacity, Apache Spark speeds up the execution of iterative jobs. This framework fits well the task of distributed computation of independence tests for **PC** algorithm.
- Second, we propose to estimate conditional independence between variables using kernel functions. Kernel functions allow to capture non-linear and complex relationships. We investigate various kernel types and prove that conventional correlation metric used in causal discovery is a special case.

To evaluate our method, we firstly experimented **MrPC** on six datasets provided in [4] to examine how a distributed algorithm could help speed up the **PC** algorithm. Experimental results showed that on the same computing infrastructure, **MrPC** can gain up to seven times faster than **PC-stable** [1], a sequential **PC** implementation. We then investigated how kernel functions work in capturing gene regulatory networks of *Escherichia coli* (*E. coli*) and *Saccharomyces cerevisiae* (*S. cerevisiae*), provided in the DREAM5 network inference challenge [5]. We found that kernel functions perform better than conventional correlation metric on both the datasets.

2 Proposed Method

2.1 Distributed Computation

The **PC** algorithm [8] has two main steps. In the first step, it learns from data a skeleton graph, which contains only undirected edges. The orientation of the undirected edges is performed in the second step to form an equivalence class of Directed Acyclic Graphs (DAGs). As the first step of the **PC** algorithm contributes to most of the computational costs, we only focus on the modification of this step in this paper.

The skeleton learning step of the **PC** algorithm starts with a complete, undirected graph and deletes recursively edges based on either marginal, $I(X, Y)$, or conditional, $I(X, Y|\mathbb{S})$, independence tests. For example, the edge between two causal factors X and Y is removed if we can find a set of other factors that does not include X and Y, and when conditioning on $\mathbb{S}$, X and Y are independent. In the worst case, the running time of the **PC** algorithm, is exponential to the number of nodes (variables), and thus it is inefficient when applying to high dimensional datasets.

In this paper, we model the dependency between two causal factors as an edge in a graph, where each node represents a causal factor. Under Spark, a MapReduce-enabled framework, to perform parallel computations of conditional independences, each edge being tested for conditional independence is considered as a single element in the *mappers* and is parallelly distributed to the *mappers* via executers (*Map* operations). The executers run the tests and return whether the input edges are independent in the current graph. The driver aggregates and summarises all the outputs from the *mappers*, updates the learning causal structures and decides the next step (*Reduce* operations).

The original **PC** algorithm [8] have the benefit of lesser number of tests, in comparison with **PC-stable** [1], as it updates the learning graph after every independence test. However, the structures returned by the original version [8] are dependent on the order of the couples of variables to be tested. To achieve order-independent structures, **PC-stable** [1] proposed to update the graph after completing all the tests at a level, which is a number of factors conditioned on. We parallel **PC-stable** [1] to use up the capacity of computing systems, their cores and all computers in the cluster, which is unused in the original approach. We summarise our parallel implementation of **PC-stable** algorithm in Algorithm 1, **MrPC** (abbreviated for **MapReduce PC**).

2.2 Kernel-Based Relation

Independence tests, either marginal or conditional, for the input variables are often conducted via correlations/partial correlations which are calculated from correlation matrices. Conventionally, a correlation matrix of factors implies a linear relationship between the factors. However, this assumption is neither always held in reality nor applied for all kinds of data distributions. Inspired by the work in [10], in this paper, we propose to calculate the correlation between

Algorithm `MrPC()`
- **Input :** A dataset where each sample is encoded by V features and the significant level α.
- **Output:** A graph G encodes the conditional independence among V features in the dataset.
- G = fully connected graph of V nodes
- $l = 0$: number of variables to be conditioned on
- **while** *True* **do**
  - *continued* = False
  - The driver ships all edges of current graph G to every executor.
  - The executors process each edge (X,Y) by the function independenceTest(X,Y) and send tuples (X,Y,independence,continued) back to the driver
  - The driver updates:
    - removes from G the edge (X,Y) with independence=True
    - sets *continued*=True if there exists a tuple with *continued*=True
  - l += 1
  - **if** *continued* = *False* **then**
    - **break**
  - **end**
- **end**
- **return**

Procedure `independenceTest(`X, Y`)`
- *independence* = False
- *continued* = False
- $\mathbb{N}$ =$neighbor(X) \setminus \{Y\}$
- **if** $|\mathbb{N}| \geq l$ **then**
  - **if** $|\mathbb{N}| > l$ **then**
    - *continued* = True
  - **end**
  - **for** $\mathbb{Z} \subseteq \mathbb{N}$ *and* $|\mathbb{Z}| = l$ **do**
    - **if** $I(X, Y|\mathbb{Z})$ **then**
      - *independence* = True
      - **break**
    - **end**
  - **end**
- **end**
- **return** X,Y,independence,continued

Algorithm 1: MrPC (MapReduce PC).

causal factors using kernel functions. Suppose that we have a dataset including N samples. Suppose that there are d different factors involved in each sample, i.e., each sample is encoded by a vector $\mathbf{f}_i = [f_{i,1}, f_{i,2}, ..., f_{i,d}] \in \mathbb{R}^d$. We can construct a matrix M with d rows and N columns for the dataset as follow,

$$M = \begin{pmatrix} f_{1,1} & f_{2,1} & \cdots & f_{N,1} \\ \cdots & \cdots & \cdots & \cdots \\ f_{1,d} & f_{2,d} & \cdots & f_{N,d} \end{pmatrix} \tag{1}$$

Based on M, we define a vector $\bar{\mathbf{f}} \in \mathbb{R}^d$ which contains the mean values of factor types over N samples and a vector $\mathbf{v} \in \mathbb{R}^d$ which contains the variance values of factor types. Then, we define a centric-normalised matrix $\hat{M}$ which is obtained by translating M by $\bar{\mathbf{f}}$ and normalising its k-th row by $\sqrt{v_k}$. In particular, we compute

$$\hat{M} = \begin{pmatrix} \frac{f_{1,1}-\bar{f}_1}{\sqrt{v_1}} & \frac{f_{2,1}-\bar{f}_1}{\sqrt{v_1}} & \cdots & \frac{f_{N,1}-\bar{f}_1}{\sqrt{v_1}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{f_{1,d}-\bar{f}_d}{\sqrt{v_d}} & \frac{f_{2,d}-\bar{f}_d}{\sqrt{v_d}} & \cdots & \frac{f_{N,d}-\bar{f}_d}{\sqrt{v_d}} \end{pmatrix} \tag{2}$$

Finally, we define a kernel-based correlation matrix C of $d \times d$ in which each element $C_{i,j}$ is the result of a kernel function K applied on rows i and j of $\hat{M}$. In particular, let $\hat{M}(i)$ and $\hat{M}(j)$ respectively denote the i-th and j-th row of $\hat{M}$, $C_{i,j}$ is calculated as,

$$C_{i,j} = K\left(\hat{M}\left(i\right), \hat{M}\left(j\right)\right) = \left\langle \Phi\left(\hat{M}\left(i\right)\right), \Phi\left(\hat{M}\left(j\right)\right)\right\rangle \tag{3}$$

where $\Phi(\mathbf{x})$ is an implicit function that maps a vector $\mathbf{x}$ in a low dimensional space $\mathcal{L}$ (for example, of d dimensions) to a high dimensional space $\mathcal{H}$ and $\langle \cdot, \cdot \rangle$ is the inner product of two vectors. There are several kernel functions proposed in the literature [3]. In this paper, we investigate common kernel functions including polynomial, sigmoidal, and Gaussian radial basis function (RBF).

Polynomial Kernel

$$K\left(\hat{M}\left(i\right), \hat{M}\left(j\right)\right) = \left\langle \hat{M}\left(i\right), \hat{M}\left(j\right)\right\rangle^p \tag{4}$$

where p is the degree of the kernel. Note that when $p = 1$, $K\left(\hat{M}\left(i\right), \hat{M}\left(j\right)\right)$ induces to the conventional correlation of two factors types i and j. This shows that the polynomial kernel is a generalised form of the conventional correlation, which only captures linear relationships. Indeed, when $p = 1$, we have,

$$K\left(\hat{M}\left(i\right), \hat{M}\left(j\right)\right) = \frac{\frac{1}{N}\sum_{k=1}^{N}\left(f_{i,k} - \bar{f}_i\right)\left(f_{j,k} - \bar{f}_j\right)}{\sqrt{v_i}\sqrt{v_j}} = Corr(i,j) \tag{5}$$

where $Corr(i,j)$ is the correlation between variables i and j.

Gaussian Radial Basis Function (RBF) Kernel

$$K\left(\hat{M}\left(i\right), \hat{M}\left(j\right)\right) = \exp\left(-\frac{\left\|\hat{M}\left(i\right) - \hat{M}\left(j\right)\right\|_2^2}{2\sigma^2}\right) \tag{6}$$

where $\|\cdot\|_2$ is the L_2- norm and σ is a user parameter set to 0.1 in our experiments.

Sigmoidal Kernel

$$K\left(\hat{M}\left(i\right),\hat{M}\left(j\right)\right)=\tanh\left(\left\langle\hat{M}\left(i\right),\hat{M}\left(j\right)\right\rangle\right) \tag{7}$$

where $\tanh\left(z\right)=\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$.

Finally, we compute the partial correlations $\rho_{i,j}$ for every pair of factors as,

$$\rho_{i,j}=\frac{C_{i,j}^{-1}}{\sqrt{C_{i,i}^{-1}C_{j,j}^{-1}}} \tag{8}$$

where C^{-1} is the inverse matrix of C.

3 Experiments

3.1 MrPC (distributed) V. PC-stable (sequential)

In the first experiment, we evaluated **MrPC** on six benchmark datasets provided in [4]. On each dataset, we also compared the running time of **MrPC** with that of the sequential approach **PC-stable** in [1]. The same parameter setting was used in both approaches, such as the significant level (α) is set to 0.01 in both settings. Also, the same computing infrastructure was used in both approaches.

Table 1. Running time (in minutes) of **MrPC** v. the sequential version (**PC-stable** [1]) on six datasets [4].

Dataset	#samples	#variables	**PC-stable**	**MrPC**	Ratio
NCI-60	47	1,190	4.57	1.66	2.75
BR51	50	1,592	31.34	10.64	2.95
MCC	88	1,380	25.73	9.63	2.67
Scerevisiae	63	5,361	79.15	22.65	3.49
Saureus	160	2,810	197.71	57.97	3.41
DREAM5-Insilico	805	1,643	1301.32	177.11	7.35

The running time needed for the **PC** algorithm by **MrPC** and the sequential approach **PC-stable** [1] is shown in Table 1. On six datasets experimented, **MrPC** performs between 2.75 and 7.35 times faster than the **PC-stable** [1]. Through experiments, we found that the gain is less on small datasets as the majority of running time is spent for Spark's initialisation, for example, locating and distributing resources to computing elements (executors). However, on larger datasets, the same amount of time and computation is spent for preprocessing but this amount is much smaller than the total running time, as the number of independence tests often increases with the size of the datasets. In summary, the larger the dataset is, the more benefit **MrPC** gains, and as shown in Table 1, the speed is improved up to seven times on DREAM5-Insilico dataset.

Table 2. Performance of different kernel functions in capturing gene regulatory networks of for *E. coli* and *S. cerevisiae* [5]. The best performances are in bold, the worst are in italics.

Kernel	AUROC	AUPR
Correlation	*0.50233*	*0.01402*
Polynomial ($p = 2$)	0.50836	0.01689
Polynomial ($p = 3$)	**0.51813**	**0.02892**
RBF	0.50349	0.01436
Sigmoidal	0.50238	0.01404

a.*E. coli* dataset.

Kernel	AUROC	AUPR
Correlation	*0.49918*	*0.01720*
Polynomial ($p = 2$)	0.50069	0.01749
Polynomial ($p = 3$)	0.50055	0.01747
RBF	**0.50102**	**0.01756**
Sigmoidal	0.49922	0.01720

b. *S. cerevisiae* dataset.

3.2 Kernel Functions

In the second experiment, we implemented **MrPC** with different kernel functions and compared these kernels with the conventional correlation computation, in building gene regulatory networks (GRN) for *E. coli* and *S. cerevisiae*. We conducted the experiment on the datasets provided in the DREAM5 network inference challenge [5], which aimed to reconstruct GRN from high-throughput data. Area under receiver operating characteristic (AUROC) and area under the precision-recall (AUPR) curves were used as performance measures [5].

We report the performance of the proposed **MrPC**with different kernels in building GRN for *E. coli* and for *S. cerevisiae* in Table 2(a) and Table 2(b) respectively. Experimental results show that, cubic polynomial kernel ($p = 3$) performs best on *E. coli* dataset in both AUROC and AUPR measures (see Table 2(a)). The second place also belongs to polynomial kernel but square polynomial kernel ($p = 2$). We found that both cubic and square polynomial kernels outperform the conventional correlation, which is proven as a case of polynomial kernels with $p = 1$. RBF kernel takes the third place while sigmoidal kernel slightly outperforms the conventional correlation. We note that this ranking is consistent in both AUROC and AUPR measures.

On *S. cerevisiae* dataset (see Table 2(b)), RBF kernel shows superior performance and achieves the first place. Cubic and square polynomial kernels perform similarly yet surpass the conventional correlation. Like *E. coli* dataset, there is slight difference in the performance of sigmoidal kernel and the conventional correlation on *S. cerevisiae* dataset. However, it is still clear to see the improvement of sigmoidal kernel compared with the conventional correlation.

4 Conclusion

Discovering causal links for observational data is an important problem with implications in causal discovery research. **PC** is a well-known tool for that task but is also a time-consuming algorithm. In an attempt to parallelise **PC**, thanks to advancements in distributed/cluster computing, we propose **MrPC** to accelerate **PC** for causal discovery in distributed systems. In addition, equipped with

MrPC, we propose to model non-linear causal relationships by using kernel functions to build gene regulatory networks from gene expression data. Experimental results on benchmark datasets show that the proposed **MrPC** is faster than sequential **PC** implementation, and kernel-based modelling outperforms conventional linear causal modelling in constructing gene regulatory networks.

References

1. Colombo, D., Maathuis, M.H.: Order-independent constraint-based causal structure learning. JMLR **15**(1), 3741–3782 (2014)
2. Dean, J., Ghemawat, S.: MapReduce: simplified data processing on large clusters. Commun. ACM **51**(1), 107–113 (2008)
3. Hofmann, T., Schölkopf, B., Smola, A.J.: Kernel methods in machine learning. Ann. Stat. **36**(3), 1171–1220 (2008)
4. Le, T., Hoang, T., Li, J., Liu, L., Liu, H., Hu, S.: A fast PC algorithm for high dimensional causal discovery with multi-core PCs. IEEE/ACM Trans. Comput. Biol. Bioinform. **16**(5), 1483–1495 (2019)
5. Marbach, D., et al.: Wisdom of crowds for robust gene network inference. Nat. Methods **9**(8), 796 (2012)
6. Shimizu, S., Hoyer, P.O., Hyvärinen, A., Kerminen, A.: A linear non-Gaussian acyclic model for causal discovery. JMLR **7**, 2003–2030 (2006)
7. Silverstein, C., Brin, S., Motwani, R., Ullman, J.: Scalable techniques for mining causal structures. Data Min. Knowl. Disc. **4**(2–3), 163–192 (2000)
8. Spirtes, P., Glymour, C.N., Scheines, R., Heckerman, D.: Causation, Prediction, and Search. MIT Press, Cambridge (2000)
9. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the USENIX Conference on Hot Topics in Cloud Computing, pp. 10 (2010)
10. Zhang, K., Peters, J., Janzing, D., Schölkopf, B.: Kernel-based conditional independence test and application in causal discovery. In: Proceedings of UAI, pp. 804–813 (2011)