Load-Balanced Parallel Constraint-Based Causal Structure Learning on Multi-Core Systems for High-Dimensional Data

Christopher Schmidt christopher.schmidt@hpi.de
Johannes Huegle johannes.huegle@hpi.de
Philipp Bode philipp.bode@student.hpi.de
Matthias Uflacker matthias.uflacker@hpi.de
Enterprise Platform and Integration Concepts
Hasso Plattner Institute, University of Potsdam
Potsdam, Germany

Editor: Thuc Duy Le, Jiuyong Li, Kun Zhang, Emre Kıcıman, Peng Cui, and Aapo Hyvärinen

Abstract

In the context of high-dimensional data state-of-the-art methods for constraint-based causal structure learning, such as the PC algorithm, are limited in their application through their worst case exponential computational complexity. To address the resulting long execution time, several parallel extensions have been developed to exploit modern multi-core systems. These extensions apply a static distribution of tasks to the execution units to achieve parallelism, which introduces the problem of load imbalance. In our work, we propose a parallel implementation that follows a dynamic task distribution in order to avoid situations of load imbalance and improve the execution time. On the basis of an experimental evaluation on real-world high dimensional datasets, we show that our implementation has a better load balancing compared to an existing parallel implementation in the context of multivariate normal distributed data. For datasets that introduce load imbalance, our dynamic task distribution approach outperforms existing static approaches by factors up to 2.4. Overall, we increase the speed up from factors of up to 27, for the static approach, to factors of up to 39 for the dynamic approach, when scaling to 80 cores compared to a non-parallel execution.

Keywords: Causal Structure Learning, Parallel computing, Constraint-based methods, High-Dimensional data, PC Algorithm

1. Introduction

Discovering causal structures from observational data is an active field of research in statistics and data mining. Understanding the causal relationships between observed variables in complex systems enables new insights and is of particular interest in the context of high-dimensional settings such as in personalized medicine or the Internet of Things. Furthermore, randomized controlled experiments, the standard method for discovering causality, oftentimes becomes infeasible due to costs and ethical concerns in these settings.

Causal graphical modeling is a recognized method for causal discovery (Pearl, 2009; Heckerman et al., 1995; Spirtes et al., 2000). In this context, causal relationships are encoded in a Directed Acyclic Graph (DAG). Two common approaches for learning these causal structures from data to derive the DAG are search and score-based and constraint-

based. The search and score-based approach evaluates a score function on all possible DAGs, which raises an NP-hard (Chickering et al., 2004). The constraint-based approach utilizes conditional independence (CI) tests to find the underlying, undirected structure of the DAG, which is also known as the adjacency search. In the worst case, this procedure is exponential to the number of variables. In a second step, the undirected edges are oriented through the repeated application of deterministic rules. Spirtes et al. proposed the PC algorithm (Spirtes et al., 2000), which runs in polynomial time for sparse graphs. Still, the algorithm's long runtime hinders its application in practice, especially in the context of high-dimensional data (Le et al., 2015a).

With recent advances in hardware technology, such as, the increasing number of cores per Central Processing Unit (CPU), multi-core and many-core systems or the adaption of Graphics Processing Units (GPUs), the relevance of parallel processing is ubiquitous. Several adaptations to constraint-based approaches, such as the PC algorithm, enabling parallel processing have been proposed (Madsen et al., 2015, 2017; Le et al., 2015a; Schmidt et al., 2018) and have shown the potential to reduce the algorithm's runtime. The approaches focus on parallel execution of the time-consuming adjacency search, phase 2 according to the template for constraint-based structure learning (Scutari, 2017) and the first required step of the PC algorithm. Some of these extensions are available in different causal graphical model learning packages, such as `pcalg` (Kalisch et al., 2012), `bnlearn` (Scutari, 2010) or `parallelPC` (Le et al., 2018).

Parallel processing requires a mapping of tasks onto execution units, e.g., CPU cores, to distribute the work. In general, the mapping techniques are classified into two categories namely static or dynamic. A naive static mapping distributes the same amount of tasks to each core. It introduces low overhead and is well suited under the assumption that the runtime of tasks is similar. Yet, load imbalances for non-uniform tasks can reduce speed-up, as the overall runtime is determined by the longest-running core.

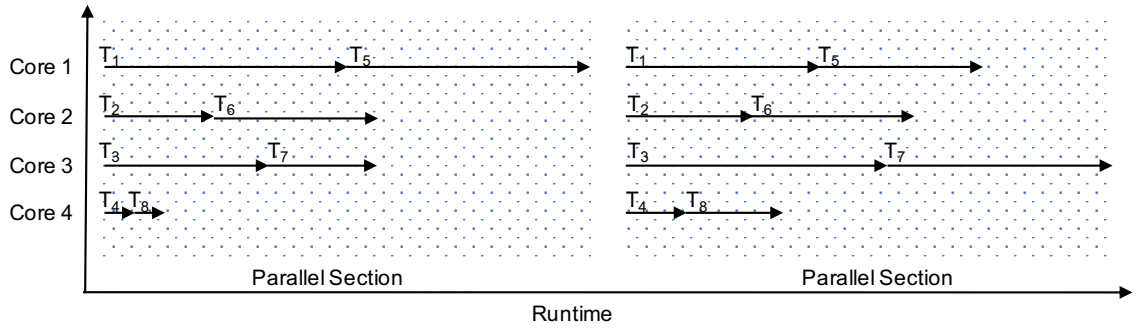

Figure 1: The impact of a naive static mapping, for the distribution of 8 non-uniform tasks T_i, $i = 1, \ldots, 8$, to 4 cores on the total runtime. Note, the total runtime depends on the longest-running cores Core 1 and 3.

This issue is visualized in Figure 1. Two subsequent parallel sections are executed with 4 cores. The naive static mapping distributes the 8 non-uniform tasks T_i, $i = 1, \ldots, 8$,in each parallel section, in a round robin fashion. This results in idle times for 3 cores while waiting on Core 1 to finish in the first parallel section and waiting on Core 3 to finish in the second parallel section. Hence, the overall runtime is the sum of the runtime of Core 1 in the first parallel section and the runtime of Core 3 in the second parallel section.

The parallel implementations of the PC algorithm, e.g., in the R-Package `pcalg` (Kalisch et al., 2012), make use of a similar naive static mapping. Given the definition of tasks for parallel execution, the number of CI tests per task is non-uniform and may lead to load imbalance and poor utilization of hardware resources in multi-core systems.

To substantiate this shortcoming, Table 1 presents the load imbalance of a naive static mapping measured on a series of real-world gene expression datasets (Marbach et al., 2012; Le et al., 2015b; H Maathuis et al., 2010). The ideal amount of CI tests per core is calculated based on the total number of CI tests divided by the number of available cores, in our experimental setup 80 and represents the perfect load balancing. The minimum, median and maximum amount of CI tests per core result from the measured distribution of tasks. Considering the maximum amount of CI tests per core, an increase of up to 70% compared to the ideal amount of CI tests per core becomes visible.

Dataset	CI Tests Per Core				Deviation Ideal
	Ideal	Minimum	Median	Maximum	to Maximum
`NCI-60`	50,218	20,531	49,955	74,145	47.65 %
`MCC`	274,378	41,526	280,310	418,179	52.41 %
`BR51`	371,203	97,388	368,701	632,104	70.29 %
`S.aureus`	2,130,386	1,713,241	2,136,229	2,502,899	17.49 %
`S.cerevisiae`	866,523	747,453	862,822	1,019,208	17.62 %
`DREAM5-Insilico`	7,384,382	4,423,809	7,376,986	10,704,566	44.96 %

Table 1: The number of CI tests executed per core, using pcalg with OpenMP enabled for parallel execution, running on 80 cores given a tuning parameter $\alpha = 0.01$.

To mitigate this load imbalance, we present a parallel implementation that uses a dynamic mapping. The implementation utilizes a centralized task queue and a number of workers, each running on a separate core, which polls tasks from the queue. We experimentally evaluate the implementation comparing the load balance with varying numbers of workers to the parallel version from the `pcalg` package. Furthermore, we investigate if a better load balance with a dynamic mapping leads to a speeds-up, despite additional overhead during runtime.

The remainder of this paper is organized as follows. Section 2 provides the necessary background on constraint-based causal structure learning algorithms, in particular, the PC algorithm. In Section 3, we discuss related work on parallel constraint-based causal structure learning. Afterward in Section 4, we present our queue-based implementation, following a dynamic mapping technique. The evaluation of the implementation is provided in Section 5, before we conclude our work in Section 6.

2. Background

In this section, we introduce some necessary terminology in the context of causal structure learning and background information about the PC algorithm, in particular on the adjacency search used for the skeleton estimation.

2.1 Graphical Model

Let a graph $\mathcal{G}$ be defined as a pair $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ consisting of a finite set of N vertices $\mathbf{V} = (V_1, \ldots, V_N)$, each representing the observed variables V_i, $i = 1, \ldots, N$, and a set of edges $\mathbf{E} \subseteq \mathbf{V} \times \mathbf{V}$. An edge $(V_i, V_j) \in \mathbf{E}$ is called directed, i.e., $V_i \to V_j$, if $(V_i, V_j) \in \mathbf{E}$ but $(V_j, V_i) \notin \mathbf{E}$. If both $(V_i, V_j) \in \mathbf{E}$ and $(V_j, V_i) \in \mathbf{E}$ the edge is called undirected, i.e., $V_i - V_j$. In this context, the skeleton $\mathcal{C}$ of a graph $\mathcal{G}$ is the result of replacing all directed edges by undirected edges.

Further, if there exists an edge (V_i, V_j) within the skeleton $\mathcal{C}$ of $\mathcal{G}$, then the two vertices V_i and V_j, are called adjacent. The adjacency set $adj(\mathcal{G}, V_i)$ of the vertex $V_i \in \mathbf{V}$ in $\mathcal{G}$ are all vertices $V_j \in \mathbf{V}$ that are directly connected to V_i by an edge in the skeleton of $\mathcal{G}$.

If all edges $\mathbf{E}$ of $\mathcal{G}$ are directed and $\mathcal{G}$ does not contain any cycle, then the graph $\mathcal{G}$ is a DAG. In the framework of causal inference, a directed edge $V_i \to V_j$ of a DAG represents a direct causal relationship of V_i to V_j (Pearl, 2009; Spirtes et al., 2000). Moreover, a DAG entails information about the conditional independencies of the vertices based on the d-separation criterion (Pearl, 2009). Using the d-separation criterion, two variables $V_i, V_j \in \mathbf{V}$ are conditionally independent given a set $\mathbf{S} \subset \mathbf{V} \setminus \{V_i, V_j\}$ if and only if the vertices V_i and V_j are d-separated by the set $\mathbf{S}$. A distribution P of the variable set $V_1, \ldots, V_N$ that satisfies the above condition is called faithful. The same CI information can be described by multiple DAGs that form a Markov equivalent class (Andersson et al., 1997). Two Markov equivalent DAGs share the same skeleton $\mathcal{C}$ and the same v-structures. A v-structure is defined as a triple (V_i, V_j, V_k) with directed edges $V_i \to V_j$ and $V_k \to V_j$ for non-adjacent vertices V_i and V_k. Moreover, the corresponding Markov equivalent class can be described uniquely by a Complete Partially Directed Acyclic Graph (CPDAG) (Chickering, 2002). A CPDAG is a partially directed acyclic graph for which all DAGs in the represented Markov equivalence class share the same directed edges and there exist two DAGs in the Markov equivalence class, one which contains the directed edge $V_i \to V_j$ and the other contains $V_j \to V_i$, for every undirected edge $V_i - V_j$. Thus, the goal is to estimate the equivalence class of the DAG $\mathcal{G}$ based on the corresponding probability distribution P. In particular, if P is faithful with respect to $\mathcal{G}$, we have the case that

$$\begin{aligned} &\text{there is an edge } V_i - V_j \text{ in the skeleton of } \mathcal{G} \\ &\quad \Leftrightarrow V_i,\ V_j \text{ are dependent given all } \mathbf{S} \subset \mathbf{V} \setminus \{V_i, V_j\}, \end{aligned} \tag{1}$$

see (Spirtes et al., 2000). Therefore, the examination of the information about the conditional independencies of the observed variables $V_1, \ldots, V_N$ enables the estimation of the skeleton $\mathcal{C}$ of the corresponding DAG $\mathcal{G}$. The estimation of the skeleton $\mathcal{C}$ by an adjacency search of the involved variables is a common first step in constraint-based algorithms for causal structure learning. Through the repeated application of deterministic orientation rules, the skeleton $\mathcal{C}$ is extended to the equivalence class of the DAG $\mathcal{G}$, e.g., see (Colombo and Maathuis, 2014; Spirtes, 2010; Kalisch and Bühlmann, 2007).

2.2 PC Algorithm

A well-known algorithm for constraint-based causal structure learning is the PC algorithm introduced by Spirtes et al. (2000). It first determines the skeleton $\mathcal{C}$ by an adjacency search and orientates the edges in a subsequent step, by orienting unshielded triples in $\mathcal{C}$ based on the examined separation set $\mathbf{S}$ and applying deterministic orientation rules. The original version of the PC algorithm depends on the order of the variable set $V_1, \ldots, V_N$. With the extension by Colombo and Maathuis (2014) an order-independent version, the PC-stable was introduced and is the basis of our work. In particular, we focus on the parallel processing of the adjacency search to determine the skeleton $\mathcal{C}$. An outline of the adjacency search is given in Algorithm 1. Starting with a complete undirected skeleton $\mathcal{C}$ the PC-stable algorithm uses CI tests with an increasing separation set $\mathbf{S}$ of adjacent vertices in order to subsequently remove edges from the skeleton $\mathcal{C}$ for which variables are determined as being independent. Hence, the algorithm only needs to query CI tests of vertices V_i and V_j given separation sets $\mathbf{S}$ with size $l = 0$ up to the maximum size of the adjacency sets of the vertices in the underlying DAG $\mathcal{G}$, i.e., up to $\max_{V_i \in \mathbf{V}} |adj(\mathcal{G}, V_i) \setminus \{V_j\}|$ (see lines 8-16 in Algorithm 1). This makes the algorithm computationally feasible even for a large number of variables and enables its application in high-dimensional settings (Kalisch and Bühlmann, 2007).

Algorithm 1 Adjacency search of PC-stable algorithm (Colombo and Maathuis, 2014)
Input: Vertex set V, tuning parameter α
Output: Estimated skeleton $\mathcal{C}$, separation sets **Sepset**

1: Start with fully connected skeleton $\mathcal{C}$ and $l = -1$
2: **repeat**
3: $l = l + 1$
4: **for all** Variables V_i in $\mathcal{C}$ **do**
5: Let $a(V_i) = adj(\mathcal{C}, V_i)$;
6: **end for**
7: **repeat**
8: Select pairs (V_i, V_j) adjacent in $\mathcal{C}$ with $|a(V_i) \setminus \{V_j\}| \geq l$
9: **repeat**
10: Choose separation set $\mathbf{S} \subseteq a(V_i) \setminus \{V_j\}$ with $|\mathbf{S}| = l$.
11: **if** $\mathrm{p}(V_i, V_j | \mathbf{S}) \leq \alpha$ **then**
12: Delete edge $V_i - V_j$ from $\mathcal{C}$;
13: Save $\mathbf{S}$ in **Sepset**;
14: **end if**
15: **until** edge $V_i - V_j$ is deleted in $\mathcal{C}$
16: or all $\mathbf{S} \subseteq a(V_i) \setminus \{V_j\}$ with $|\mathbf{S}| = l$ have been chosen
17: **until** all adjacent vertices V_i, and V_j in $\mathcal{C}$ such that
18: $|a(V_i) \setminus \{V_j\}| \geq l$ have been considered
19: **until** each adjacent pair V_i, V_j in $\mathcal{C}$ satisfy $|a(V_i) \setminus \{V_j\}| \leq l$
20: **return** $\mathcal{C}$, **Sepset**

For every level $l = 0, \ldots, \max_{V_i \in \mathbf{V}} |adj(\mathcal{G}, V_i) \setminus \{V_j\}|$ the adjacency sets $a(V_i) = adj(\mathcal{C}, V_i)$ of variables V_i with respect to the current skeleton $\mathcal{C}$ are computed and stored (see lines

4-6). Hence, at each level l, the algorithm records the edges that need to be removed but deletes these edges only when entering the next level $l + 1$. Note, this allows for an order-independent implementation of the original PC algorithm as proven by Colombo et al. (Colombo and Maathuis, 2014), and remains true for parallel versions of the PC algorithm, e.g., see (Le et al., 2015a).

First, for $l = 0$ all pairs of vertices $V_i, V_j \in \mathbf{V}$ are tested for conditional independence given an empty separation set $\mathbf{S} = \emptyset$. Given an overall tuning parameter α, an independence test suitable for the data distribution of the variables is applied and it is examined whether $\mathrm{p}(V_i, V_j | \mathbf{S}) = \mathrm{p}(V_i, V_j | \emptyset) \leq \alpha$, or not. If the two variables V_i, V_j are determined to be independent, the edge $V_i - V_j$ is deleted from $\mathcal{C}$ and the empty set $\emptyset$ is stored in **Sepset** (see lines 11-13 in Algorithm 1). Once, all pairs of vertices have been tested, the algorithm proceeds to the next step with $l = 1$.

When $l = 1$, the algorithm applies the CI tests for vertices V_i and V_j that are still adjacent in the skeleton $\mathcal{C}$. Therefore, it is now examined whether $\mathrm{p}(V_i, V_j | \mathbf{S}) \leq \alpha$ given the separation set $\mathbf{S} \subset adj(\mathcal{C}, V_i) \setminus V_j$ for subsets of size $l = 1$. Hence, if the variables V_i and V_j are found to be conditionally independent given the corresponding separation set $\mathbf{S}$, the edge $V_i - V_j$ is removed, and $\mathbf{S}$ is stored in **Sepset**. The algorithm increments l by one, once all pairs of adjacent vertices V_i and V_j of the current skeleton $\mathcal{C}$ are tested.

This step is repeated until l reaches the maximum size of the adjacency sets of the vertices in the underlying DAG $\mathcal{G}$. The resulting skeleton $\mathcal{C}$ and the separation sets **Sepset** provide the basis for the application of deterministic orientation rules in order to extend $\mathcal{C}$ to the corresponding CPDAG, e.g., see Colombo and Maathuis (2014); Spirtes (2010); Kalisch and Bühlmann (2007).

2.3 Parallel Adjacency Search

A parallel version of the adjacency search, outlined in Algorithm 1, requires the definition of tasks, and mapping for the distribution of tasks to the executing cores. High parallel efficiency is achieved by defining tasks that are independent of each other requiring no synchronization and communication. A mapping for the tasks either occurs statically or dynamically during runtime. A static mapping incurs little overhead, yet is prone to load imbalance. It is well suited for uniform tasks with the same amount of work. A dynamic mapping introduces overhead at runtime but achieves better load balance for non-uniform tasks with an unknown amount of work.

For the parallel execution of the adjacency search, suitable tasks are defined as the calculation of the CI information for each pair of vertices $V_i, V_j \in \mathbf{V}$ within a single level l (Le et al., 2015a; Scutari, 2017), (see lines 7 - 17 in Algorithm 1). Under the assumption that the adjacency sets for all variables are calculated prior to the parallel execution (see lines 4 - 6 in Algorithm 1), the tasks are independent of each other within a single level l.

For level $l = 0$ this definition of a task implies uniform tasks, under the assumption that the runtime of the applied test for conditional independence is the same for all pairs of vertices $V_i, V_j \in \mathbf{V}$. For example, this assumption holds true for multivariate normal distributed data where CI tests may be based on the sample correlation coefficient. In contrast, this assumption does not necessarily hold for categorical data where χ^2 tests are

applied. In this context, the calculation time of a single χ^2 test varies with the number of categories and category combinations of the variables $V_i, V_j \in \mathbf{V}$.

In subsequent levels $l \geq 1$, vertices V_i and V_j adjacent in the skeleton $\mathcal{C}$, in the current level l, are tested for conditional independence given a separation set $\mathbf{S}$. The separation set $\mathbf{S}$ is drawn from the set of adjacent vertices $adj(\mathcal{C}, V_i)$, until either the edge between the vertices V_i and V_j is deleted or all separation sets have been considered. Hence, the tasks are no longer uniform, as a varying number of separation sets $\mathbf{S}$ may be considered for each pair of vertices V_i and V_j.

3. Related Work

A critical obstacle for the application of constraint-based causal structure learning in high-dimensional data settings, in particular for the PC algorithm, is the algorithm's long runtime. In the worst case, it is exponential to the number of variables for a given dataset. Therefore, optimizations have been proposed to reduce the number of CI tests, for example by ranking the edges (Cano et al., 2008) or replacing the CI tests with Bayesian statistical tests (J. Abellán and M. Gómez-Olmedo and S. Moral, 2006). Furthermore, parallel implementations have been introduced to efficiently utilize modern multi-core hardware or accelerators (Madsen et al., 2015, 2017; Le et al., 2015a; Schmidt et al., 2018). The majority of parallel implementations focus on the order-independent extension of the PC algorithm, called PC-stable (Colombo and Maathuis, 2014).

In particular, a parallel version of the PC algorithm is introduced by Le et al. (2015a) and implemented in the R-package `ParallelPC`. Using real-world gene expression datasets they show that their parallel implementation achieves a performance improvement compared to the non-parallel PC-stable algorithm from the R-package `pcalg`. The implementation utilizes the CPU cores to process the CI tests within level l in parallel. It uses the R-package `parallel` to statically distribute all edges, keeping the CI tests for a single edge on the same core. According to the authors, distributing CI tests for the same edge to different cores is not efficient. Depending on the dataset, the chosen approach may lead to an unbalanced workload distribution among the processing units, as shown in Figure 1.

Madsen et al. (2017) propose an implementation for a parallel PC algorithm, which targets a shared memory computer, as well as, a computer cluster with distributed memory. Their approach utilizes the concept of Balanced Incomplete Block Designs (Madsen et al., 2014) to distribute the computations of CI tests to cores in a shared memory system and to assign computations to worker nodes in a distributed system. The work focuses on multinomial distributions and handles higher-order CI tests through ranking edges according to a test score with the goal to perform the most promising CI tests first.

With the goal to exploit parallel compute capabilities of a GPU, Schmidt et al. (2018) ported the skeleton discovery of the PC-Stable to the accelerator. In their implementation, they rely on the static parallel execution strategy proposed by Le et al. (2015a) and adapt it to utilize GPU specific hardware characteristics, such as shared memory and the Single Instruction Multiple Threads (SIMT) execution model. While they show significant speed-up compared to existing parallel versions, their approach is limited in several ways. First, it focuses only on CI tests with low order, given that only a small number of higher-order CI tests are required for the given datasets. Second, they only consider data with a Gaussian

distribution and do not provide implementation or parallelization strategies for other data distributions, such as multinomial or mixed distributions. Further, load imbalance is not considered for their GPU-accelerated implementation.

The R-package `bnlearn` implements several constraint-based causal structure learning algorithms. The implementations follow a general framework for parallel constraint-based learning presented by Scutari (2017). Similar to other approaches tasks are statically distributed to the parallel execution units, potentially leading to load imbalance. It is argued that under the assumption of a sparse Bayesian network the impact of the load imbalance is small. Further, the author shows that the implementation scales well for up to 8 parallel execution units.

The R-package `pcalg` (Kalisch et al., 2012) provides a parallel implementation for the adjacency search given a multivariate normal distribution. The parallel version is implemented in C++ using openMP (Dagum and Menon, 1998). By using openMP it enables the use of pragmas in the source code, e.g., `#pragma omp parallel for`, to define loops that are executed in parallel. While openMP allows defining different mappings for tasks to cores, these are not specified in the implementation. Therefore the default static mapping is used, following a block distribution. In this case, the tasks are grouped into evenly sized blocks distributing the same amount of tasks to each core.

All of the mentioned parallel approaches follow a naive static mapping accepting the potential load imbalance. In contrast, we propose a dynamic distribution of tasks to workers, each running on a separate core, in order to address the load imbalance. Ideally, this improves the execution time of constraint-based causal structure learning algorithms, in particular, the PC-Stable.

4. A Load-Balanced Parallel Adjacency Search for the PC Algorithm

In the following, we describe our parallel implementation of the adjacency search utilizing a centralized task queue to achieve better load balancing. Our implementation follows the outline of the PC-stable algorithm but splits the algorithm into three separate parts. In the first part, all CI tests for level $l = 0$ are conducted. Given that no separation set $\mathbf{S}$ is required in level $l = 0$, the tasks may be uniform. In that case, dynamic mapping yields no benefit. Therefore, in the case of multivariate normal distributed data, we follow the existing parallel implementation of the `pcalg` and distribute the tasks in level $l = 0$ in the same static manner using OpenMP. For the remaining levels, $l \geq 1$ we implement one part executed by the producer outlined in Algorithm 2 and a second part, which is executed by multiple worker instances sketched in Algorithm 3.

The producer is responsible for the centralized queue utilized to implement the dynamic mapping of tasks to cores. In our implementation, we utilize a lock-free, high-throughput queue [1] in order to accommodate a large number of tasks, given many vertices or a dense skeleton $\mathcal{C}$. At first, the producer fills the queue with tasks, meaning all pairs of vertices (V_i, V_j), which have to be tested for conditional independence in the current level l, (see line 8 in Algorithm 2). Once the queue is filled, the producer creates a working copy $\mathcal{C}_{copy}$ of the skeleton $\mathcal{C}$. This ensures that removing any edges in the skeleton by one worker has no impact on any subsequent tasks within the current level l. Workers, processing the tasks in

1. https://github.com/cameron314/concurrentqueue

Algorithm 2 Producer
Input: Vertex set V, number of worker N_{Worker}, tuning parameter α
Output: Estimated skeleton $\mathcal{C}$, separation sets **Sepset**

Start with fully connected skeleton $\mathcal{C}$ and $l = 0$
repeat
  $l = l + 1$
  for all Variables V_i in $\mathcal{C}$ **do**
    Let $a(V_i) = adj(\mathcal{C}, V_i)$;
  end for
  repeat
    Fill Queue with pairs (V_i, V_j) adjacent in $\mathcal{C}$ with $|a(V_i) \setminus \{V_j\}| \geq l$
  until all adjacent V_i, and V_j in $\mathcal{C}$ with $|a(V_i) \setminus \{V_j\}| \geq l$ have been considered
  Create working copy $\mathcal{C}_{copy}$ of skeleton $\mathcal{C}$;
  Start N_{Worker} Worker with $(V, \mathcal{C}, \mathcal{C}_{copy}, \mathbf{Sepset}, l, \alpha)$;
  Wait until Queue is empty;
  $\mathcal{C} = \mathcal{C}_{copy}$;
until each adjacent pair V_i, V_j in $\mathcal{C}$ satisfy $|a(V_i) \setminus \{V_j\}| \leq l$
return $\mathcal{C}$, **Sepset**

the queue, are started according to the number of workers N_{Worker} provided. Each worker is mapped onto a separate core. After all the tasks in the queue have been processed, the results are merged. Under the assumption of a shared memory system and the independence of tasks, all workers have been working on the same shared working copy $\mathcal{C}_{copy}$. Therefore, the merge is a pointer redirection from $\mathcal{C}_{copy}$ to $\mathcal{C}$. This ensures that the correct skeleton is used in the subsequent level. The producer continues with the next level l, until a level l is reached for which all pairs of vertices (V_i, V_j) satisfy $|a(V_i) \setminus \{V_j\}| \leq l$. Analogous to the PC-stable, the resulting skeleton $\mathcal{C}$ is used as the basis for the application of deterministic orientation rules to extend $\mathcal{C}$ to the corresponding CPDAG.

Algorithm 3 Worker
Input: Vertex set V, current version of $\mathcal{C}$, working copy of $\mathcal{C}_{copy}$, separation sets **Sepset**, level l, tuning parameter α

repeat
  Take task with pair (V_i, V_j) from Queue;
  Let $a(V_i) = adj(\mathcal{C}, V_i)$;
  repeat
    Choose separation set $\mathbf{S} \subseteq a(V_i) \setminus \{V_j\}$ with $|\mathbf{S}| = l$.
    if $\mathrm{p}(V_i, V_j | \mathbf{S}) \leq \alpha$ **then**
      Delete edge $V_i - V_j$ from $\mathcal{C}_{copy}$;
      Set $\mathbf{Sepset}(i, j) = \mathbf{Sepset}(j, i) = \mathbf{S}$;
    end if
  until edge $V_i - V_j$ is deleted in $\mathcal{C}_{copy}$ or all $\mathbf{S} \subseteq a(V_i) \setminus \{V_j\}$ with $|\mathbf{S}| = l$ have been chosen
until Queue is empty

The workers are responsible to process the tasks available in the centralized queue. In order to achieve load balancing, each worker itself is responsible to poll a new task, once it has finished its current task. The worker polls new tasks from the central queue until there are no more tasks available. While processing a task, the worker follows the same steps as described in Subsection 2.2 for the PC-stable, with one exception. In the case that the pair of vertices (V_i, V_j) is found to be independent, the worker deletes the edge in the working copy $\mathcal{C}_{copy}$ of the skeleton. Our C++ implementation of the worker utilizes the standard library function `std::next_permutation` to choose the separation set **Sepset** for level $l \geq 2$. At the core, it uses the same CI test implementation for a multivariate normal distribution that is used in the Rcpp (Eddelbuettel and Balamuta, 2017) extension of the R-Package `pcalg`.

5. Evaluation

Within the following experiments, we compare the use of a naive static mapping to the use of a dynamic mapping when executing the adjacency search of the PC-stable algorithm (Colombo and Maathuis, 2014) in parallel. In particular, we measure the difference in the work that is done per core for each level l, varying the number of cores. We assume that a dynamic mapping is better suited for the parallel tasks of the PC-algorithm because it improves the load balance. Hence, the maximum number of tasks per core should be closer to the ideal distribution, i.e., the equal distribution of CI tests. Yet, the dynamic mapping introduces overhead at runtime. To ensure that the overhead does not outweigh the performance gain, we also measure and compare the runtime of both approaches.

Experimental Setup The experiments are executed on an enterprise-grade server with 2 Intel ® Xeon ® Gold 6148 CPU with 20 cores each, leading to 80 cores including hyper-threading. The server is equipped with 1.5 TB of RAM, allowing to keep all data in memory during the execution of the experiments. The operating system is an Ubuntu 18.04 with GCC version 6.5 supporting OpenMP version 4.5 and R version 3.4.4. In order to focus the experiments on a comparison of a static mapping to a dynamic mapping, we try to eliminate any influencing factors, e.g., different programming languages, i.e., if comparing to ParallelPC (Le et al., 2015a) or Tetrad (Scheines et al., 1998) or different hardware, i.e., if comparing to a GPU-accelerated implementation (Schmidt et al., 2018). Hence, in the experiments, we use the implementation of the `pcalg` [2] version 2.6, built with OpenMP enabled representing a naive static mapping and our implementation of a dynamic mapping based on a centralized task queue [3] described in Section 4. Both implementations use the same implementation of the CI test for Gaussian data. Other data distributions are supported neither in the `C++` extension of the `pcalg` nor in our implementation. For measurements of the runtime, we utilize the `high_resolution_clock` from the `C++` standard library `chrono`. In particular, we measure the runtime for each level l separately. When reporting numbers regarding the balance of work per core, we added counters per worker to the parallel sections in both implementations. During all experiments, there is no additional

2. https://github.com/cran/pcalg/
3. https://github.com/philipp-bode/lock-free-pc/tree/measurements

Dataset	$\|\mathbf{V}\|$	Observations	$\|\mathbf{E}\|$	Number of CI tests per level $l=1$	$l=2$	$l>=3$
NCI-60	1,190	47	530	2,760,231	56,347	286
MCC	1,380	88	643	11,155,607	3,560,149	2,212
BR51	1,592	50	478	22,495,684	246,919	491
S.aureus	2,810	160	2,058	113,106,125	22,843,938	13,386
S.cerevisiae	5,361	63	2,086	54,598,391	355,867	117
DREAM5-Insilico	1,643	805	5,929	105,082,379	336,939,542	129,125,057

Table 2: Characteristics of gene expression datasets used in the experiments. Note $|\mathbf{E}|$ and the number of CI tests per level are determined with a tuning parameter $\alpha = 0.01$.

load on the system. If not stated differently we report the median value of 10 measured executions.

Datasets Our experimental evaluation is based on gene expression datasets (Marbach et al., 2012; Le et al., 2015b; H Maathuis et al., 2010), which are the foundation in the evaluation of the `Parallel-PC` algorithm by Le et al. (2015a).

The characteristics of the high-dimensional datasets are described in Table 2. The number of variables in the datasets ranges from 1,190 to 5,361. Note that the number of observations for the datasets is small. Given its impact on the acceptance of CI tests and our chosen value for the tuning parameter $\alpha = 0.01$ (Kalisch and Bühlmann, 2007), no higher level l with large separation sets $\mathbf{S}$ is reached. Hence, for all datasets, with the exception of the `DREAM5-Insilico` dataset, most CI tests are conducted in level $l = 1$ and the maximum level is between $l = 4$ and $l = 5$. For the `DREAM5-Insilico` dataset, the majority of CI tests are conducted in level $l = 2$ followed by level $l = 3$ and $l = 1$. The maximum level that is reached during the adjacency search is level $l = 13$. The number of CI tests for level $l = 0$ are not reported. For level $l = 0$ both implementations use the same static mapping with a naive block-based distribution, which is beneficial for the uniform tasks in the Gaussian case. Therefore level $l = 0$ is also not considered in any of the following measurements.

5.1 Experiments on Load Balance

With the following experiments, we examine the load balance of our implemented parallel adjacency search based on a centralized queue. As mentioned in the Introduction 1, the overall runtime of the algorithm is the result of the longest-running core within each level l. Therefore, we focus on the cores that conducted the maximum number of CI tests within each level l. We calculate the sum of these maxima and report the percentage compared to an ideal distribution of CI tests. The ideal distribution is calculated as the quotient of the total number of CI tests divided by the number of cores. In Figure 2, we compare these percentages for the adjacency search of the centralized queue implementation with the ones for the adjacency search of the `pcalg` package with a naive static task distribution on several gene expression datasets. During the experiment, all cores available on our test system were utilized. Note, for the naive static distribution no differences can be measured as the same mapping of tasks to cores occurs in each execution.

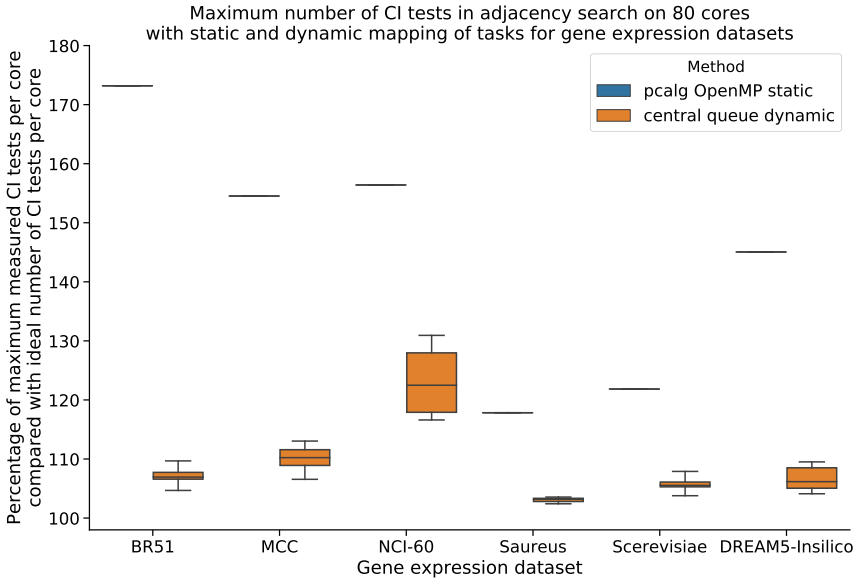

Figure 2: The load balance for different gene expression datasets, on 80 cores, comparing dynamic and static task mapping strategies. Note, values closer to 100 are better.

The results show that the centralized queue implementation has a lower maximum number of CI tests in the longest-running cores across all level l in the adjacency search. This is due to an improved load balance, given the dynamic task mapping. With the exception of the `NCI-60` dataset, the maximum number of CI tests is less than 13% off the ideal value, compared with up to 73% for the static task distribution, e.g., for the dataset `BR51`. For `NCI-60` we account the smaller improvement for the centralized queue implementation to the fact that fewer CI tests are conducted within each level and that the tasks are more balanced.

In the following, we examine the maximum number of CI tests in the longest-running cores over all level l with a varying number of cores. Thereby we aim to evaluate if the centralized queue also works in other settings, i.e., with fewer cores. We start with 2 cores and increase the number of cores as multiples of 2, including numbers that are specific to the hardware setup, i.e., the number of cores per socket, e.g., 20, 40, 80. We select two datasets, `BR51`, and `DREAM5-Insilico` based on characteristics presented in Table 2. The dataset `BR51` has a small number of conducted CI tests, whereas `DREAM5-Insilico` has the largest number of conducted CI tests and reaches the highest level l compared to the other gene expression datasets. The percentages of the maximum number of CI tests compared to an ideal distribution per core are shown in Figure 3 for `BR51` on the top and `DREAM5-Insilico` on the bottom.

For the dataset `BR51` the maximum number of CI tests in the longest-running cores is improved using the centralized queue once tasks are distributed to 4 or more cores. Using 16 or more cores the maximum number of CI tests in the longest-running cores increases strongly. Hence, the imbalance in the load increases as well. For the centralized queue, the value remains steady when the number of cores is increased. Similar behavior of

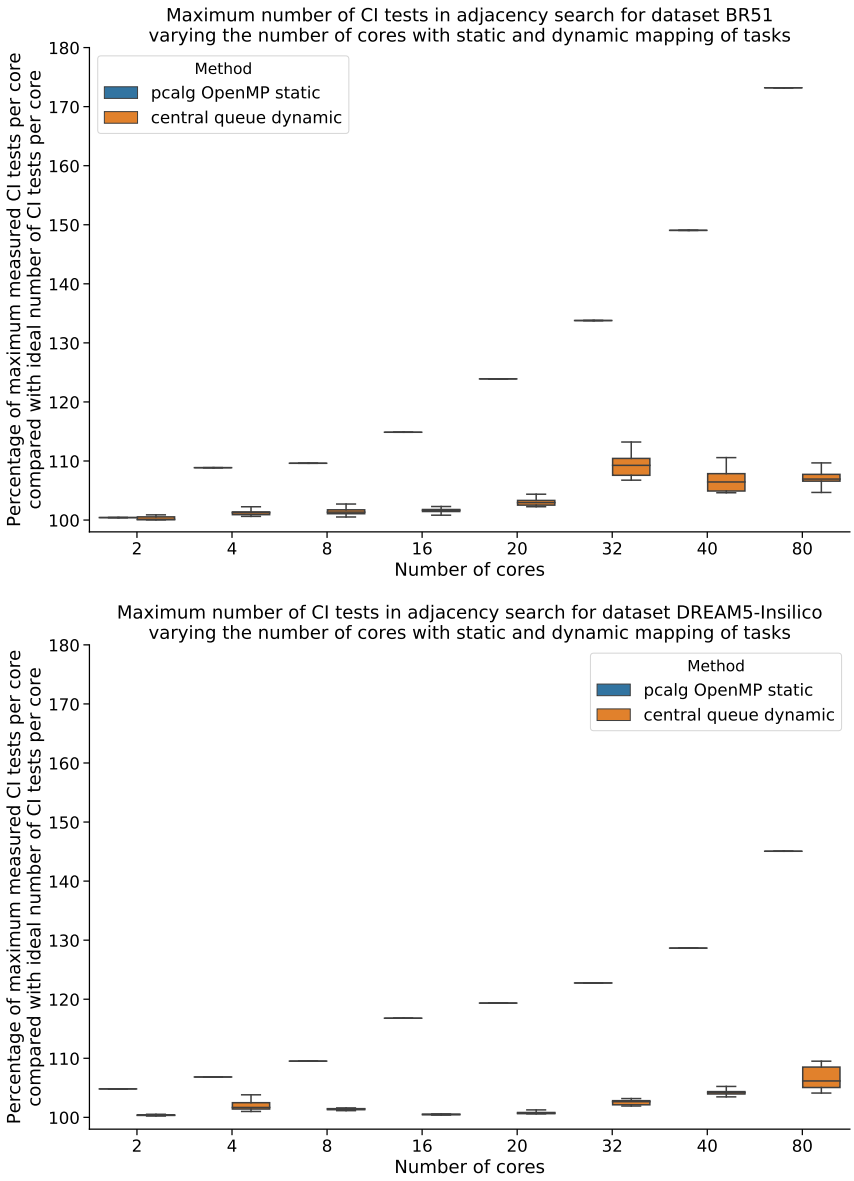

Figure 3: The load balance for an increasing number of cores, comparing dynamic and static task mapping strategies on dataset BR51 (top) and DREAM5-Insilico (bottom).

the centralized queue is visible for dataset `DREAM5-Insilico`. Yet, the reduced maximum number of CI tests in the longest-running cores for the centralized queue compared to the naive static mapping is already visible for 2 cores. Further, the imbalance for the naive static mapping is smaller, compared to the dataset `BR51`. In general, our centralized queue approach achieves a better load balancing and thereby reduces the number of CI tests in the longest-running cores of the adjacency search in multi-core systems, in particular for a larger number of cores. A setup, in which a naive static task distribution as employed by default in the `pcalg` package suffers from load imbalance.

5.2 Experiments on Runtime

In order to verify that a better load balance improves the overall performance of the adjacency search for high-dimensional datasets, we investigate the runtimes for the two mapping strategies. We base our measurements on the same gene expression datasets as before.

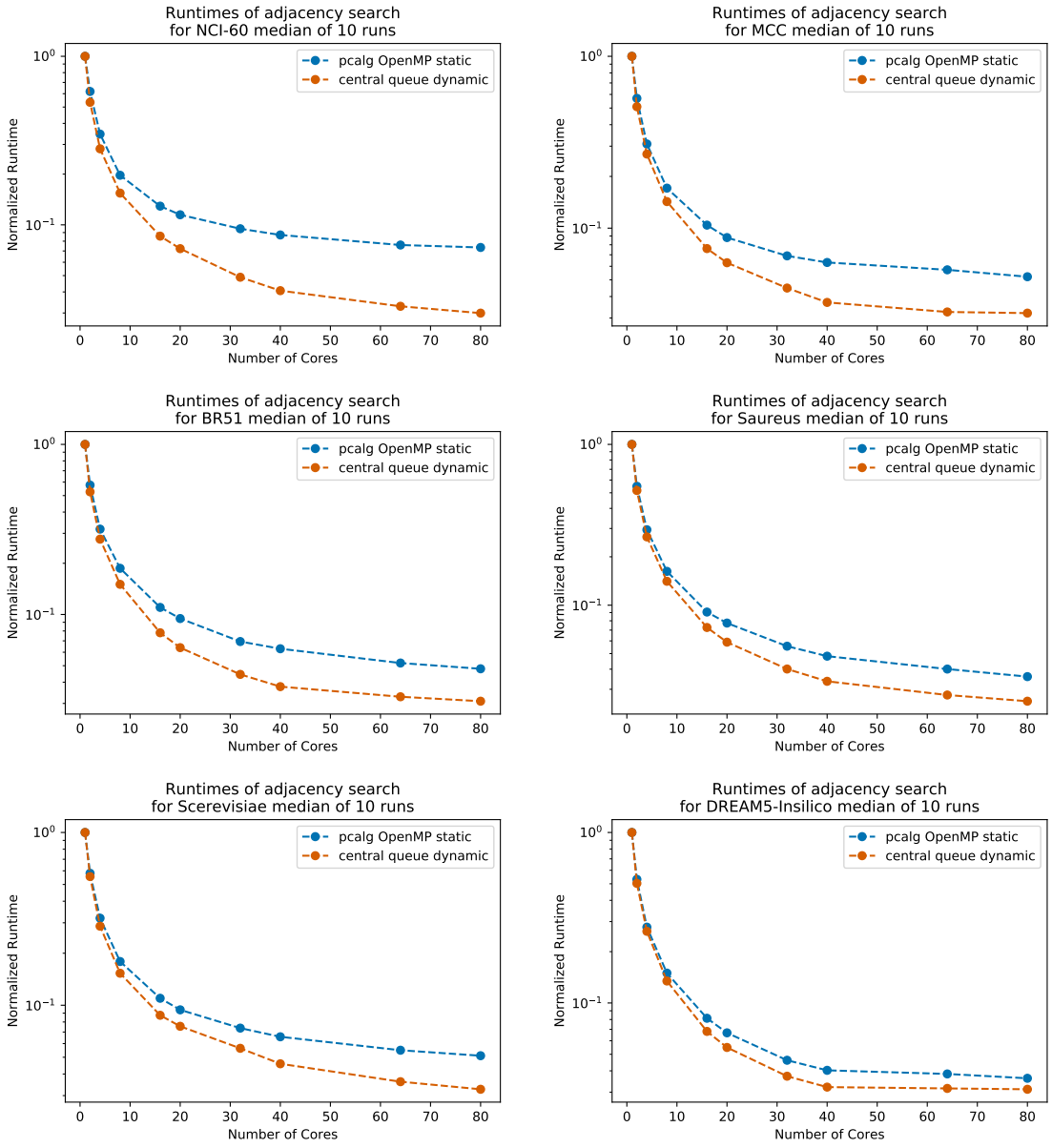

Figure 4: Normalized runtimes for the adjacency search on gene expression datasets with a varying number of cores, comparing pcalg with OpenMP static to the centralized queue implementation.

In Figure 4, we report the normalized runtimes with a varying number of cores. Note, the leftmost dots in each subfigure correspond to single-threaded execution on one core. We observe an improved normalized runtime for the centralized queue compared to the OpenMP static approach in all measured combinations of the number of cores and high-dimensional datasets. The performance gap between the two strategies increases with an increasing number of cores for each dataset and is the largest for settings with at least 40 cores. We observed a maximum speed up, by factors between 1.25 for 40 cores on dataset `DREAM5-Insilico` to 2.45 for 80 cores on dataset `NCI-60`. When using a few cores, i.e., 2 or 4, the difference in performance is marginal. Further, the performance difference varies across the dataset, which we account for the underlying graphical structure, which is unique for each dataset. We investigated factors, such as the total number of CI tests conducted or the number of CI tests per level l, and did not observe a significant influence on the performance difference for the high-dimensional datasets. Moreover, we observe that compared to a non-parallel version we achieve speed up, by factors up to 39 with the centralized queue approach, compared to speed up, by factors up to 27 for the OpenMP static approach. While we are able to improve the performance, room for improvement is still possible as the parallel efficiency reaches only up to 50% when scaling to 80 cores. We observe that the measured runtimes follow the law of diminishing returns stating that adding more workers to the parallel section leads to decreasing improvements with each new worker. This result matches previous experimental observations from Scutari (2017).

6. Discussion and Conclusion

In this paper, we proposed a parallel implementation of the adjacency search used for the derivation of the skeleton of graph $\mathcal{G}$, a substantial part in constraint-based causal structure learning algorithms, such as the PC-algorithm. The implementation dynamically distributes tasks to the executing cores in a multi-core system. Thereby, we address the load imbalance present in existing parallel implementations of the adjacency search, which employ a naive static task distribution that does not account for non-uniform tasks. We experimentally evaluate our approach and compare it to an existing solution. Hereby, we focus on high-dimensional datasets, for which the dynamic distribution is thought to improve the performance (Scutari, 2017), and on the scalability with the number of cores available for parallel execution. We observed that the dynamic distribution reduces the longest-running cores across all levels l within the adjacency search, hence improving the load balance. This results in a maximum speed-up of up to a factor 2.45 on the evaluated real-world gene expression datasets compared to a naive static task distribution. When the tasks are distributed to a small number of cores, the load imbalance is not a strong issue for the naive static task distribution and speed-up is marginal. Comparing to a non-parallel version we achieve a speed-up, by factors up to 39, compared to factors up to 27 for a naive static distribution.

In our work, we have considered CI tests on Gaussian data only. For future work, an extension to other distributions, i.e., multinomial distributions, or nonparametric tests could be of interest. Depending on the implementation of the CI test, e.g., using a contingency table, all observations related to the vertices V_i and V_j may have to be processed. Therefore, we assume that the amount of work per task increases, leading to higher load imbalance,

making a dynamic distribution more relevant. Yet, in that setting data locality has to be considered, as more data is accessed while processing a single CI test, exceeding hardware cache sizes of caches shared by multiple cores.

Summarizing, we conclude that the dynamic distribution of tasks is well suited for parallel execution of the adjacency search on large multi-core, or even many-core systems, in high-dimensional datasets given a multivariate Gaussian distribution.

Acknowledgments

The authors would like to thank Hendrik Raetz, Frederic Schneider and Nils Thamm for helping with the implementation of the parallel adjacency search utilizing a centralized queue.

References

Steen A. Andersson, David Madigan, and Michael D. Perlman. A Characterization of Markov Equivalence Classes for Acyclic Digraphs. *The Annals of Statistics*, 25(2):505–541, 1997. ISSN 00905364. URL http://www.jstor.org/stable/2242556.

A. Cano, M. Gómez-Olmedo, and S. Moral. A Score Based Ranking of the Edges for the PC Algorithm. In Manfred Jaeger and Thomas D. Nielsen, editors, *Proceedings of the Fourth European Workshop on Probabilistic Graphical Models*, pages 41–48, 2008.

David Maxwell Chickering. Learning Equivalence Classes of Bayesian-network Structures. *J. Mach. Learn. Res.*, 2:445–498, March 2002. ISSN 1532-4435. doi: 10.1162/153244302760200696. URL https://doi.org/10.1162/153244302760200696.

David Maxwell Chickering, David Heckerman, and Christopher Meek. Large-Sample Learning of Bayesian Networks is NP-Hard. *J. Mach. Learn. Res.*, 5:1287–1330, December 2004. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1005332.1044703.

Diego Colombo and Marloes H. Maathuis. Order-independent Constraint-based Causal Structure Learning. *J. Mach. Learn. Res.*, 15(1):3741–3782, January 2014. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=2627435.2750365.

Leonardo Dagum and Ramesh Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Comput. Sci. Eng.*, 5(1):46–55, January 1998. ISSN 1070-9924. doi: 10.1109/99.660313. URL https://doi.org/10.1109/99.660313.

Dirk Eddelbuettel and James Joseph Balamuta. Extending *R* with *C++*: A Brief Introduction to *Rcpp*. *PeerJ Preprints*, 5:e3188v1, aug 2017. ISSN 2167-9843. doi: 10.7287/peerj.preprints.3188v1. URL https://doi.org/10.7287/peerj.preprints.3188v1.

Marloes H Maathuis, Diego Colombo, Markus Kalisch, and Peter Bühlmann. Predicting causal effects in large-scale systems from observational data. *Nature Methods*, 7:247–8, 04 2010. doi: http://dx.doi.org/10.1038/nmeth0410-247.

David Heckerman, Dan Geiger, and David M. Chickering. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. *Machine Learning*, 20(3):197–243, Sep 1995. ISSN 1573-0565. doi: 10.1023/A:1022623210503. URL https://doi.org/10.1023/A:1022623210503.

J. Abellán and M. Gómez-Olmedo and S. Moral. Some Variations on the PC Algorithm. In *Proceedings of the Third European Workshop on Probabilistic Graphical Models (PGM' 06)*, pages 1–8, 2006.

Markus Kalisch and Peter Bühlmann. Estimating High-Dimensional Directed Acyclic Graphs with the PC-Algorithm. *J. Mach. Learn. Res.*, 8:613–636, May 2007. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1248659.1248681.

Markus Kalisch, Martin Mächler, Diego Colombo, Marloes H. Maathuis, and Peter Bühlmann. Causal Inference Using Graphical Models with the R package pcalg.

Journal of Statistical Software, Articles, 47(11):1–26, 2012. ISSN 1548-7660. doi: 10.18637/jss.v047.i11. URL https://www.jstatsoft.org/v047/i11.

Thuc Le, Tao Hoang, Jiuyong Li, Lin Liu, Huawen Liu, and Shu Hu. A fast PC algorithm for high dimensional causal discovery with multi-core PCs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 02 2015a.

Thuc Duy Le, Lin Liu, Junpeng Zhang, Bing Liu, and Jiuyong Li. From miRNA regulation to miRNA-TF co-regulation: computational approaches and challenges. *Briefings in Bioinformatics*, 16(3):475–496, 2015b. doi: 10.1093/bib/bbu023. URL +http://dx.doi.org/10.1093/bib/bbu023.

Thuc Duy Le, Taosheng Xu, Lin Liu, Hu Shu, Tao Hoang, and Jiuyong Li. ParallelPC: An R Package for Efficient Causal Exploration in Genomic Data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 207–218, Cham, 2018. Springer International Publishing. ISBN 978-3-030-04503-6.

Anders L. Madsen, Frank Jensen, Antonio Salmerón, Martin Karlsen, Helge Langseth, and Thomas D. Nielsen. A New Method for Vertical Parallelisation of TAN Learning Based on Balanced Incomplete Block Designs. In Linda C. van der Gaag and Ad J. Feelders, editors, *Probabilistic Graphical Models*, pages 302–317, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11433-0.

Anders L. Madsen, Frank Jensen, Antonio Salmerón, Helge Langseth, and Thomas D. Nielsen. Parallelisation of the PC Algorithm. In *Proceedings of the 16th Conference of the Spanish Association for Artificial Intelligence on Advances in Artificial Intelligence - Volume 9422*, pages 14–24, New York, NY, USA, 2015. Springer-Verlag New York, Inc. ISBN 978-3-319-24597-3. doi: 10.1007/978-3-319-24598-0_2. URL http://dx.doi.org/10.1007/978-3-319-24598-0_2.

Anders L. Madsen, Frank Jensen, Antonio Salmerón, Helge Langseth, and Thomas D. Nielsen. A Parallel Algorithm for Bayesian Network Structure Learning from Large Data Sets. *Know.-Based Syst.*, 117(C):46–55, February 2017. ISSN 0950-7051. doi: 10.1016/j.knosys.2016.07.031. URL https://doi.org/10.1016/j.knosys.2016.07.031.

Daniel Marbach, James C. Costello, Robert Küffner, Nicole M. Vega, Robert J. Prill, Diogo M. Camacho, Kyle R. Allison, Manolis Kellis, James J. Collins, Andrej Aderhold, Gustavo Stolovitzky, and et al. Wisdom of crowds for robust gene network inference. *Nature Methods*, 9(8):796–804, 8 2012. ISSN 1548-7091.

Judea Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, New York, NY, USA, 2nd edition, 2009. ISBN 052189560X, 9780521895606.

Richard Scheines, Peter Spirtes, Clark Glymour, Christopher Meek, and Thomas Richardson. The TETRAD Project: Constraint Based Aids to Causal Model Specification. *Multivariate Behavioral Research*, 33(1):65–117, 1998. doi: 10.1207/s15327906mbr3301_3.

Christopher Schmidt, Johannes Huegle, and Matthias Uflacker. Order-independent Constraint-based Causal Structure Learning for Gaussian Distribution Models Using

GPUs. In *Proceedings of the 30th International Conference on Scientific and Statistical Database Management*, SSDBM '18, pages 19:1–19:10, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-6505-5. doi: 10.1145/3221269.3221292. URL http://doi.acm.org/10.1145/3221269.3221292.

Marco Scutari. Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software, Articles*, 35(3):1–22, 2010. ISSN 1548-7660. doi: 10.18637/jss.v035.i03. URL https://www.jstatsoft.org/v035/i03.

Marco Scutari. Bayesian Network Constraint-Based Structure Learning Algorithms: Parallel and Optimized Implementations in the bnlearn R Package. *Journal of Statistical Software, Articles*, 77(2):1–20, 2017. ISSN 1548-7660. doi: 10.18637/jss.v077.i02. URL https://www.jstatsoft.org/v077/i02.

Peter Spirtes. Introduction to Causal Inference. *J. Mach. Learn. Res.*, 11:1643–1662, August 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1756006.1859905.

Peter Spirtes, Clark N Glymour, and Richard Scheines. *Causation, Prediction, and Search*. MIT press, 2000.