Making Queries Tractable on Big Data with Preprocessing

Making Queries Tractable on Big Data with Preprocessing

(through the eyes of complexity theory)

Wenfei Fan
Informatics, University of Edinburgh &
RCBD and SKLSDE Lab, Beihang University
wenfei@inf.ed.ac.uk

Floris Geerts
University of Antwerp
floris.geerts@ua.ac.be

Frank Neven
Hasselt University &
transnational University of Limburg
frank.neven@uhasselt.be

Abstract

A query class is traditionally considered *tractable* if there exists a polynomial-time (PTIME) algorithm to answer its queries. When it comes to big data, however, PTIME algorithms often become infeasible in practice. A traditional and effective approach to coping with this is to preprocess data off-line, so that queries in the class can be subsequently evaluated on the data efficiently. This paper aims to provide a formal foundation for this approach in terms of computational complexity. (1) We propose a set of Π-*tractable* queries, denoted by $\Pi\mathsf{T}^0_\mathsf{Q}$, to characterize classes of queries that can be answered in *parallel poly-logarithmic time* (NC) after PTIME preprocessing. (2) We show that several natural query classes are Π-tractable and are feasible on big data. (3) We also study a set $\Pi\mathsf{T}_\mathsf{Q}$ of query classes that can be effectively converted to Π-tractable queries by re-factorizing its data and queries for preprocessing. We introduce a form of NC reductions to characterize such conversions. (4) We show that a natural query class is *complete* for $\Pi\mathsf{T}_\mathsf{Q}$. (5) We also show that $\Pi\mathsf{T}^0_\mathsf{Q} \subset \mathsf{P}$ unless $\mathsf{P} = \mathsf{NC}$, *i.e.,* the set $\Pi\mathsf{T}^0_\mathsf{Q}$ of all Π-tractable queries is *properly contained* in the set P of all PTIME queries. Nonetheless, $\Pi\mathsf{T}_\mathsf{Q} = \mathsf{P}$, *i.e.,* all PTIME query classes can be *made* Π-tractable via proper re-factorizations. This work is a step towards understanding the tractability of queries in the context of big data.

1. Introduction

Challenges introduced by big data suggest that we depart from the traditional view on tractability. As found in most textbooks (*e.g.,* [1, 33]), a class of queries is traditionally considered *tractable* if there exists an algorithm for answering its queries in time bounded by a polynomial (PTIME) in the size of the input, *i.e.,* a database and a query. In other words, a class of queries is *feasible* from a theoretical perspective if its worst-case time complexity is PTIME, while a class is considered difficult to solve when it is NP-hard.

In practice, however, PTIME queries do not always serve as a good yardstick for tractability. This is more evident in the context of big data. Consider a dataset D of 1 PetaByte (PB, 10^{15} bytes). Assuming the fastest Solid State Drives (SSD) with disk scanning speed of 6GB/s [38], a linear scan of D takes 166,666 seconds; that is, 46 hours, or 1.9 days!

A popular traditional yet effective approach to coping with this is to preprocess a database by, *e.g.,* building indices, which are then used to accelerate query evaluation. When the data is mostly static or when the indices can be maintained efficiently, this preprocessing step can be considered as an off-line process with a *one-time cost*. Further, as the indices typically serve to answer a multitude of queries, this one-time cost can often be ignored, and the actual computation cost is measured in terms of the online evaluation of the queries with the indices. The need for such preprocessing becomes even more evident when querying big data.

Example 1: Consider a class $\mathcal{Q}_1$ of point-selection queries. A query $Q_1 \in \mathcal{Q}_1$ on a relation D is to find whether there exists a tuple $t \in D$ such that $t[A] = c$, where A is an attribute of D and c is a constant. A naive evaluation of Q_1 would require a linear scan of D. In contrast, we can first build a B^+-tree on the values of the A column in D [34], in a one-time preprocessing step off-line. Then we can answer *all queries* $Q_1 \in \mathcal{Q}_1$ on D in $O(\log|D|)$ time, by capitalizing on such indices. That is, we no longer need to conduct a linear scan of the data when processing *each* query in $\mathcal{Q}_1$. When D consists of 1PB of data, we can get the results in seconds with the indices rather than 1.9 days. □

This example demonstrates that characterizing a query class as being in PTIME does not tell us much about its feasibility on big data. Instead, one needs to revise the traditional notion of tractability to provide a dichotomy between those queries that can be made feasible on big data after appropriate preprocessing and those for which preprocessing does not help. In addition to building indices as shown in the example, many more preprocessing strategies have proven effective and are being widely used in practice. These include (*i*) query-preserving compression schemes that preserve the answers to a class of queries rather than preserving the data itself [16, 24, 31, 32], (*ii*) building views such that queries can be answered using the views without accessing the original big data [1, 23, 30]; and (*iii*) bounded incremental algorithms [35] that, after evaluating queries once on the original data (as preprocessing), incrementally evaluate the queries in response to the changes to the data such that the cost of query evaluation is a function of the size of the changes, rather than the size of the original big data [15, 37].

While it has been a common practice to speed up query evaluation with preprocessing, a formal framework is not yet in pace to answer a number of fundamental questions. What query classes can be considered tractable in this setting? How large is the set of all query classes that are tractable with preprocessing? Is every class of PTIME

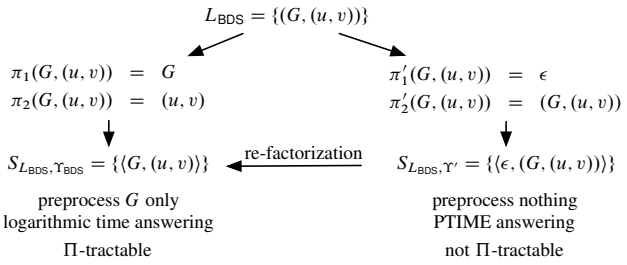

Figure 1: Two factorizations Υ_{BDS} and Υ' of the decision language L_{BDS} for BDS

queries tractable on big data when preprocessing is allowed? If not, can it be *made tractable* by means of efficient transformations such that appropriate preprocessing is possible and hence, it is feasible to answer its queries on big data? Is there a complete problem for the set of all queries that can be made tractable with preprocessing, just like our familiar NP-complete problems for the class NP [33]?

Contributions. This paper aims to provide a formal foundation, in terms of computational complexity, for studying the tractability of query classes in the context of big data with preprocessing. We note that preprocessing is just one of the approaches to handling big data. Several models have been proposed that depart from the traditional computation model by distributing data and resources across multiple computing nodes, and measuring complexity in terms of coordination or communication cost instead of time (see, e.g., [2, 3, 28, 29]). These approaches are orthogonal to the model proposed here (see Section 2 for a detailed discussion).

(1) Π-tractable queries. We propose a notion of Π-tractable queries. Consider a query class $\mathcal{Q}$ in which each query Q operates on some database D. We say that $\mathcal{Q}$ is Π-*tractable* if there exists a polynomial $\Pi(\cdot)$ such that for any database D, we can convert D into D' in time bounded by $\Pi(|D|)$, and moreover, for all queries $Q \in \mathcal{Q}$, $Q(D)$ can be computed as $Q(D')$ in NC, *i.e.,* in parallel polylog-time in the sizes $|D|$ and $|Q|$ with polynomially many processors.

We denote by $\Pi\mathsf{T}^0_{\mathsf{Q}}$ the set of all Π-tractable query classes.

That is, after a preprocessing step on D in PTIME, *all queries* in $\mathcal{Q}$ defined on D can be subsequently answered in polylog-time by leveraging parallel computation. We will justify the choice of complexity classes (PTIME for preprocessing and NC for online query evaluation) shortly.

When we study the complexity class of all Π-tractable queries, we consider *w.l.o.g.* Boolean query languages $\mathcal{Q}$ [1], as usual. That is, for each query $Q \in \mathcal{Q}$ and database D, the answer $Q(D)$ of Q in D is either true or false.

(2) Case studies. As examples, we show that several natural query classes are Π-tractable. In addition to relational selection queries discussed in Example 1, $\Pi\mathsf{T}^0_{\mathsf{Q}}$ includes reachability queries on graphs, minimum range queries on static arrays [18], queries for computing lowest common ancestors in trees and directed acyclic graphs (DAGs) [5], and a class of graph queries for breadth-depth search [21]. We also present strategies for preprocessing data, by means of query-preserving compression, query answering using views and bounded incremental computation, such that queries can be subsequently evaluated efficiently on big data.

(3) Making queries Π-tractable via NC reductions. While some query classes $\mathcal{Q}$ are not Π-tractable, $\mathcal{Q}$ can actually be transformed to a Π-tractable query class by means of *re-factorizations*, which re-partition the data and query parts of $\mathcal{Q}$ and identify a data set for preprocessing, such that after the preprocessing, its queries can be subsequently answered in parallel polylog-time, *i.e., $\mathcal{Q}$* can be *made* Π-tractable.

Similarly, a *decision problem* can be *made* Π-*tractable* if each of its instances can be *factorized* into a pair $\langle D, Q\rangle$, where D denotes data and Q is a query in a Boolean query language $\mathcal{Q}$, such that the answer to the instance is true if and only if $Q(D)$ is true and moreover, $\mathcal{Q}$ is Π-tractable.

Intuitively, a factorization identifies a data part D from a problem instance for preprocessing. After D is preprocessed, the query part Q of the instance can be answered efficiently.

Example 2: Consider Breadth-Depth Search (BDS) [21]:

- Input: An undirected graph $G = (V, E)$ with a numbering on the nodes, and a pair (u, v) of nodes in V.
- Question: Is u visited before v in the breadth-depth search of G induced by the vertex numbering?

A breadth-depth search starts at a node s and visits all its children, pushing them onto a stack in the reverse order induced by the vertex numbering as the search proceeds. After all of s's children are visited, the search continues with the node on the top of the stack, which plays the role of s.

Figure 1 shows two factorizations of the decision language L_{BDS} associated to BDS: Υ_{BDS} that identifies G as the data-part and (u, v) as the query part; and Υ' that treats both G and (u, v) as query part, leaving nothing to be preprocessed. The definition of factorizations will be given in Section 3. We will see later that Υ_{BDS} makes BDS Π-tractable. □

We denote the set of all decision problems (resp. query classes) that can be made Π-tractable as $\Pi\mathsf{T}_{\mathsf{P}}$ (resp. $\Pi\mathsf{T}_{\mathsf{Q}}$). These are important since after all, what practitioners want to know is whether a query class or a problem can be properly factorized and be made Π-tractable.

To characterize transformations that convert a problem (query class) to a Π-tractable counterpart, we define a form of many-one reductions, denoted by $\leqslant^{\mathsf{NC}}_{fa}$, which are in NC, *i.e.,* also in parallel polylog-time. We show that such reductions are *compatible* with $\Pi\mathsf{T}_{\mathsf{P}}$: if $A \leqslant^{\mathsf{NC}}_{fa} B$ and B is in $\Pi\mathsf{T}_{\mathsf{P}}$, then so must be A; similarly for $\Pi\mathsf{T}_{\mathsf{Q}}$. Moreover, they are *transitive*, *i.e.,* if $A \leqslant^{\mathsf{NC}}_{fa} B$ and if $B \leqslant^{\mathsf{NC}}_{fa} C$, then $A \leqslant^{\mathsf{NC}}_{fa} C$. These NC reductions allow us to reduce our problems to another that we know how to solve. Furthermore, they help us determine whether a problem can be made Π-tractable.

(4) A complete problem for $\Pi\mathsf{T}_{\mathsf{P}}$. We identify a natural problem, namely, the Breadth-Depth Search problem (BDS) (see [21]), and show that BDS is *complete* for $\Pi\mathsf{T}_{\mathsf{P}}$ under $\leqslant^{\mathsf{NC}}_{fa}$. That is, BDS is one of the "hardest" problems in $\Pi\mathsf{T}_{\mathsf{P}}$ to which all problems in $\Pi\mathsf{T}_{\mathsf{P}}$ can be transformed via NC reductions $\leqslant^{\mathsf{NC}}_{fa}$. This yields an effective method to determine whether a given problem can be made Π-tractable: if the problem can be reduced to BDS via proper factorization of its instances, then it can be made Π-tractable.

Better still, we show that all query classes in P can be *made* Π-*tractable* via $\leqslant^{\mathsf{NC}}_{fa}$ reductions, where P is the set of all query classes that can be evaluated in PTIME. That is, although it may be infeasible to evaluate a class $\mathcal{Q}$ of PTIME queries on big data without preprocessing, queries in $\mathcal{Q}$ can actually be transformed to Π-tractable queries by means of re-factorizations and appropriate preprocessing.

(5) Separation of $\Pi\mathsf{T}^0_\mathsf{Q}$ *and* P. To clarify the difference between $\Pi\mathsf{T}^0_\mathsf{Q}$ and P, we introduce another form of NC reductions, referred to as F-reductions $\leqslant^{\mathsf{NC}}_F$. F-reductions transform a query class $\mathcal{Q}$ to another class $\mathcal{Q}'$, and the data sets of $\mathcal{Q}$ to the data sets of $\mathcal{Q}'$, *without re-factorizations*. We show that $\mathsf{NC} \subseteq \Pi\mathsf{T}^0_\mathsf{Q} \subseteq \mathsf{P}$, but $\Pi\mathsf{T}^0_\mathsf{Q} \neq \mathsf{P}$ unless $\mathsf{P} = \mathsf{NC}$, which is a longstanding open problem [21]. That is, while all queries in $\Pi\mathsf{T}^0_\mathsf{Q}$ are in PTIME and all NC queries are Π-tractable, *not all* PTIME query classes are Π-tractable or can be reduced via $\leqslant^{\mathsf{NC}}_F$ to any Π-tractable query class, unless $\mathsf{P} = \mathsf{NC}$.

These results tell us that $\Pi\mathsf{T}^0_\mathsf{Q}$ is a proper subset of P (unless $\mathsf{P} = \mathsf{NC}$). Nonetheless, many practical query classes can be made Π-tractable via proper factorizations.

Justification. This work is just one step towards developing a guidance for us to search for feasible solutions to query processing on big data. It aims to incite interest in the study of tractable queries on big data. Below we provide justification and intuition for the definition of $\Pi\mathsf{T}^0_\mathsf{Q}$.

(1) We allow a PTIME preprocessing step in $\Pi\mathsf{T}^0_\mathsf{Q}$ for the following reasons. (a) PTIME plays the historical role of tractable query classes [22], and moreover, is robust and well-studied [21]. (b) As remarked earlier, preprocessing is a *one-time* price and is performed *off-line*. (c) The restriction to PTIME ensures that the result of the preprocessing step is bounded by a polynomial and therefore, rules out unreasonably large indices. Of course, there are always settings in which PTIME (or even linear time) might be too steep a price to pay, even when the computation is only performed once. For those applications one might want to require preprocessing to be performed in NC or in MapReduce [28] or its variants thereof (see Section 2). If we choose NC, then $\Pi\mathsf{T}^0_\mathsf{Q}$ coincides with NC, in which many useful query classes cannot be expressed. Since MapReduce is not yet as robust or well-understood as PTIME, we opt here for PTIME to focus on the main properties of query answering with preprocessing, and defer restricted versions of our model to future work.

(2) For the query processing step in $\Pi\mathsf{T}^0_\mathsf{Q}$, we envisage a complexity class that, on one hand, is large enough to model logarithmic searches in indices or accommodate $\mathsf{polylog\text{-}time}$, and on the other hand, can be regarded feasible on big data. For this reason, we consider parallel $\mathsf{polylog\text{-}time}$, that is, the class NC. As shown in Example 1, parallel $\mathsf{polylog\text{-}time}$ is feasible on big data. Even though alternatives exist, *e.g.,* LogP [10] and BSP [40], NC remains to be the most prevalent model capturing parallel-time complexity [21] that is independent of any particular PRAM architecture. Moreover, like PTIME, NC is robust and well-understood. Furthermore, NC is one of the few parallel complexity classes whose connections with classical sequential complexity classes have been extensively studied (see, *e.g.,* [21]).

(3) We focus on databases D that are either static or allow efficient incremental preprocessing. After a database D is preprocessed and yields D', D may be updated by change ΔD. It may be too costly to preprocess $D \oplus \Delta D$ again starting from scratch. Instead, we assume incremental preprocessing of $D \oplus \Delta D$ in response to ΔD, *i.e.,* by computing $\Delta D'$ such that the outcome of processing $D \oplus \Delta D$ is the same as $D' \oplus \Delta D'$. In practice, ΔD is typically small. When ΔD is small, the change $\Delta D'$ to D' is often small as well, and is much less costly to find than to conduct the entire preprocessing of $D \oplus \Delta D$. We use incremental preprocessing to avoid paying the price of the PTIME complexity of the entire preprocessing process. Moreover, we advocate bounded incremental algorithms [35] to maintain preprocessed data (see Section 4). if such algorithms exist. A more comprehensive account of dynamic data is deferred to future work.

Organization. Section 2 reviews related work. Section 3 introduces $\Pi\mathsf{T}^0_\mathsf{Q}, \Pi\mathsf{T}_\mathsf{P}$ and $\Pi\mathsf{T}_\mathsf{Q}$, followed by several natural classes of $\Pi\mathsf{T}^0_\mathsf{Q}$ in Section 4. Section 5 defines NC reductions $\leqslant^{\mathsf{NC}}_{fa}$. Section 6 shows that BDS is complete for $\Pi\mathsf{T}_\mathsf{P}$. Section 7 proposes the notion of F-reductions, and shows that unless $\mathsf{P} = \mathsf{NC}$, $\Pi\mathsf{T}^0_\mathsf{Q} \subset \mathsf{P}$. Finally, Section 8 identifies open issues in connection with Π-tractable query classes.

2. Background and Related Work

This work is related to prior work on complexity classes within P, complexity models of MapReduce and its variants, and complexity classes defined in terms of preprocessing.

P and NC. The complexity class P consists of all decision problems that can be solved by a deterministic Turing machine in polynomial time (PTIME), *i.e.,* in $n^{O(1)}$ time, where n is the size of the input. A number of problems have been shown P-complete, *i.e.,* they are in P, and each problem in P can be reduced to them by using NC reductions (see [21] for a survey of P-complete problems). In this paper we also use P to denote the set of all PTIME query classes.

Several complexity classes within P have been well studied, notably those defined by sublinear space bounds.

- The class NL consists of all decision problems that can be solved by a nondeterministic Turing machine that uses space bounded by $O(\log n)$.
- The class $\mathsf{polylog\text{-}space}$ consists of all decision problems solvable with workspace bounded by $O(\log^{O(1)} n)$, by a deterministic or nondeterministic Turing machine.

It is known that $\mathsf{NL} \subseteq \mathsf{polylog\text{-}space} \cap \mathsf{P}$, but $\mathsf{P} \neq \mathsf{polylog\text{-}space}$ (see, *e.g.,* [26], for details).

The parallel complexity class NC, known as Nick's Class, consists of all decision problems that can be solved by taking $O(\log^{O(1)} n)$ time on a PRAM (parallel random access machine) with $n^{O(1)}$ processors (see, *e.g.,* [21,26]). It has been shown that there are natural query classes that capture NC over ordered relational databases [39]. A problem in NC is generally considered *highly parallel feasible*, *i.e.,* can be efficiently solved on a parallel computer [21]. It is known that $\mathsf{NL} \subseteq \mathsf{NC} \subseteq \mathsf{polylog\text{-}space}$, and $\mathsf{NC} \subseteq \mathsf{P}$. However, *a major open question* is whether $\mathsf{NC} = \mathsf{P}$, which is widely viewed as an analogy to the question whether $\mathsf{P} = \mathsf{NP}$ [21,26].

NC reductions are *compatible with* P: if problem A is NC reducible to B and B is in P, then A is also in P.

Parallel and distributed processing. In the database community, the term big data is often associated with parallel and distributed query processing. We next discuss work related to complexity models for distributed query evaluation. We refer to [7] for a more system-oriented overview.

It is observed in [29] that circuits and PRAM may not be accurate for parallel systems; it puts forward the massively parallel (MP) model of computation to better capture computation with a vast number of servers (in the order of tens of thousands), in a setting when the key bottleneck is the number of synchronization steps or rounds. For this reason, the MP model disregards the complexity of computation lo-

cal to each server, and only takes the number of global synchronization steps into account. The latter is sometimes also referred to as *coordination complexity*, a term introduced by [25]. The focus of [29] is to identify which fragments of conjunctive queries can be evaluated in a single round.

MapReduce [12] is a popular programming paradigm for handling big data. In this framework, [3] studies the evaluation of join queries, and takes the amount of communication, measured as the sum of the sizes of the input to reducers, as a complexity measure. Evaluation of transitive closure and datalog queries in MapReduce has been investigated in [2,4].

Observe that the prior work aforementioned falls within a larger program that also investigates which query classes can be considered tractable within various models of computation. Our work focuses on preprocessing and therefore, can be seen as orthogonal to these approaches.

There has also been recent work on simulating PRAM via MapReduce [28], which shows that a large class of NC algorithms can be implemented in the MapReduce framework, and moreover, if an NC algorithm takes t time, than its corresponding MapReduce counterpart takes $O(t)$ MapReduce rounds. There have also been attempts to revise the PRAM model by requiring $\log\, n$ processors instead of $n^{O(1)}$ [13].

Preprocessing. As remarked earlier, it is common for database people to preprocess data by building indices, creating views and compressing the data, among others. It has also been used to define complexity classes (*e.g.,* [8, 11, 19]).

A notion of compilable classes is proposed in [8]. For a complexity class C, a problem is said to be *compilable to* C if each of its instances can be partitioned into a fixed part x and a varying part y, and moreover, after preprocessing x by a poly-size function $f(\cdot)$, solving $\langle f(x), y\rangle$ is in class C. It differs from this work in the following. (1) The focus of [8] has been on intractable classes C, namely, NP and beyond. As a result, there exist complete problems for C and moreover, any C-complete problem leads to a problem complete for the compilable class to C under PTIME reductions. In contrast, we consider NC for query answering, and reductions for establishing complete problems for NC are not yet settled. For instance, when NC reductions are used, all problems in NC reduce to a trivial problem, which is not very helpful when studying the complete problem for $\Pi\mathsf{T}_\mathsf{P}$ (under $\leqslant_F^{\mathsf{NC}}$). (2) There is no limit on the time needed by the preprocessing function $f(\cdot)$ of [8], except that the size of the "compiled structure" $f(x)$ is bounded by a polynomial of $|x|$. Here $f(\cdot)$ could be even *non-recursive*, a rather impractical setting. In contrast, we require that the preprocessing step of Π-tractable queries is in PTIME.

In parameterized complexity theory [19], fixed-parameter tractability can be characterized in terms of kernelization, which reduces a given instance of a parameterized problem in PTIME to an instance whose size is bounded in terms of the parameter alone and does not depend on the size of the original instance. Here kernelization can be viewed as *preprocessing*. It differs from our work as follows. (1) Kernelization is defined on the parameter of an intractable problem, which captures the main source of the intractability of the problem. In contrast, we preprocess *the data* of Π-tractable queries by, *e.g.,* building auxiliary structures such as indices and views in addition to compressing the data, such that the queries can then be evaluated on the data efficiently. (2) When the parameter of a problem is not assumed fixed, solving the problem is not in PTIME after kernelization. In contrast, after preprocessing of the data of Π-tractable queries, the queries can be evaluated in NC.

Pseudo sublinear-time algorithms aim to solve problems in sublinear-time [11], after appropriate preprocessing. The study of sublinear-time algorithms has focused on property testing via approximation and randomization (see [11, 36] for surveys), which we do not allow here. Moreover, complexity classes and complete problems studied in this work are not considered by the prior work. On the other hand, when a problem is not in $\Pi\mathsf{T}_\mathsf{P}$, we may develop Π-tractable approximate or randomized algorithms for it.

Preprocessing has also proven effective in handling NP instances in cryptographic applications, via compression [24]. It is also related to partial evaluation, which is widely used in compiler generation, code optimization and dataflow evaluation (see [27] for a survey). Given a function $f(s, d)$ and part of its input s, partial evaluation conducts the part of $f(\cdot)$'s computation that depends only on s, and generates a partial answer, *i.e.,* a residual function $f'(\cdot)$ that depends on the as yet unavailable input d. The generation of $f'(\cdot)$ can also be viewed as preprocessing.

3. Tractable Queries on Big Data

In this section, we first formally define Π-tractable query classes ($\Pi\mathsf{T}_\mathsf{Q}^0$). We then present decision problems and query classes that can be made Π-tractable ($\Pi\mathsf{T}_\mathsf{P}$ and $\Pi\mathsf{T}_\mathsf{Q}$).

We start with some notations.

Notations. Following the convention of complexity theory [33], we assume a finite alphabet Σ of symbols to encode both data and queries. A database can be encoded as a string $D \in \Sigma^*$ just like in a Turing machine, with necessary delimiters; similarly for a query Q. The length of a string $x \in \Sigma^*$ is denoted by $|x|$. Hence $|D|$ (resp. $|Q|$) denotes the size of a database D (resp. a query Q).

A *language S of pairs* is a subset of $\Sigma^* \times \Sigma^*$. We use S to encode a class $\mathcal{Q}$ of Boolean queries such that for each $\langle D, Q\rangle \in S$, Q is a query in $\mathcal{Q}$, D is a database on which Q is defined, and $Q(D)$ is *true*. In other words, S can be considered as a binary relation such that $\langle D, Q\rangle \in S$ if and only if $Q(D)$ is true. We refer to S as *the language for* $\mathcal{Q}$.

To be consistent with the complexity classes of decision problems, we consider Boolean queries in this work. For instance, $\mathcal{Q}$ may be the class $\mathcal{Q}_1$ of Boolean point selection queries given in Example 1. This does not lose generality: given a non-Boolean query Q' and a database D, one can write a Boolean query Q to determine, given a tuple t, whether $t \in Q'(D)$. There is a natural analogy from this to the connection between search problems and decision problems in complexity theory (see, *e.g.,* [21]).

We say that a language S of pairs is *in complexity class* C if it is in C to decide whether a pair $\langle D, Q\rangle \in S$. For instance, C may be the sequential complexity class P or the parallel complexity class NC, among other things.

Π-tractable queries. Using languages of pairs to represent queries, we are now ready to define Π-tractable classes of queries, denoting tractable queries with preprocessing.

Definition 1: A language S of pairs is Π-*tractable* if there exist a PTIME preprocessing function $\Pi : \Sigma^* \to \Sigma^*$ and a language S' of pairs such that for all $D, Q \in \Sigma^*$,

- $\langle D, Q\rangle \in S$ if and only if $\langle \Pi(D), Q\rangle \in S'$, and
- S' is in NC.

We say that a class $\mathcal{Q}$ of queries is Π-*tractable* if S is Π-tractable, where S is the language of pairs for $\mathcal{Q}$. We use $\Pi\mathsf{T}^0_{\mathsf{Q}}$ to denote the set of all Π-tractable query classes. □

Intuitively, function $\Pi(\cdot)$ preprocesses data D and generates another structure $D' = \Pi(D)$, in PTIME. After this, for *all queries* $Q \in \mathcal{Q}$ that are defined on D, $Q(D)$ can be answered by evaluating $Q(D')$ in parallel polylog-time in the sizes $|D'|$ and $|Q|$. In other words, all the queries in the set $\mathcal{Q}_D = \{Q \mid Q \in \mathcal{Q}, Q \text{ is defined on } D\}$ can then be answered in D' in NC. Note that $|D'| \leq p(|D|)$ for a polynomial $p(\cdot)$.

The preprocessing function $\Pi(\cdot)$ may compress D, or build auxiliary structures such as indices without reducing the size of D, or do both, in an *off-line* PTIME computation. No matter what it does to D, after the preprocessing step, queries in $\mathcal{Q}$ can then be answered in parallel polylog-time.

To simplify the discussion, we allow preprocessing of the data but keep the queries unchanged. One may consider a more general setting by incorporating a *query rewriting function* $\lambda : \mathcal{Q} \to \mathcal{Q}'$, and revise Definition 1 such that (1) $\langle D, Q\rangle \in S$ if and only if $\langle \Pi(D), \lambda(Q)\rangle \in S'$, and (2) S' is in NC. Then as long as $\lambda(\cdot)$ is a PTIME computable function, it is still feasible to answer queries of $\mathcal{Q}$ on big data. We defer the study of this more general setting to future work.

Example 3: Recall the query class $\mathcal{Q}_1$ given in Example 1. The language S_1 for $\mathcal{Q}_1$ is a set of pairs $\langle D, (A, c)\rangle$, where D is a relation, A is an attribute of D and c is a constant, such that there exists a tuple $t \in D$ with $t[A] = c$. As shown in Example 1, $\mathcal{Q}_1$ is Π-tractable: we may preprocess D by building B^+-trees on attributes in D in PTIME. After this, for any (A, c), whether there exists $t \in D$ such that $t[A] = c$ can be decided in $O(\log|D|)$ time by using the indices.

As another example, consider a class $\mathcal{Q}_2$ of reachability queries. Given a directed graph G, a query $Q_2 \in \mathcal{Q}_2$ is to determine whether there exists a path from s to t in G, where s and t are nodes in G. This class of queries corresponds to the Graph Accessibility Problem (GAP), which is NL-complete with log-space reductions (cf. [26]). The language S_2 for $\mathcal{Q}_2$ is a set of pairs $\langle G, (s, t)\rangle$, such that there exists a path from s to t in G. Recall that NL $\subseteq$ NC, and hence $\mathcal{Q}_2$ is Π-tractable. Better still, off-line preprocessing helps us here: we may precompute a matrix that records the reachability between all pairs of nodes in G, and then answer all queries $Q_2 \in \mathcal{Q}_2$ defined on G in $O(1)$ time by using the matrix. Hence after the preprocessing we need constant time to answer each query in $\mathcal{Q}_2$, instead of NL. □

Making problems Π-tractable. We next extend the notion of Π-tractability to decision problems. Recall that a decision problem can be represented as a language $L \subseteq \Sigma^*$ that contains string encoding of its instances, such that for any instance $x \in \Sigma^*$ of the problem, $x \in L$ if and only if the solution to x is true [21]. In the sequel we use L to denote a decision problem and a language interchangeably.

We now want to define when a decision problem can be regarded as Π-tractable. That is, we want to find out whether preprocessing can help us make it feasible to solve the decision problem on big data. We answer this question by treating decision problems as languages of pairs, for which the notion of Π-tractability is already in place (Definition 1). To this end, we need a notion of factorization, to identify *which part* of the problem may be subject to preprocessing.

We say that a language L can be *factored* if there exist three NC computable functions $\pi_1(\cdot)$, $\pi_2(\cdot)$ and $\rho(\cdot,\cdot)$ such that for all $x \in L$, $\rho(\pi_1(x), \pi_2(x)) = x$. We refer to

- $\Upsilon = (\pi_1, \pi_2, \rho)$ as a *factorization* of L,
- $S_{(L,\Upsilon)} = \{\langle \pi_1(x), \pi_2(x)\rangle \mid x \in L\}$ as *the language of pairs* for (L, Υ);
- $L_{(D,\Upsilon)} = \{\pi_1(x) \mid x \in L\}$ as the *data set* of (L, Υ), and
- $L_{(Q,\Upsilon)} = \{\pi_2(x) \mid x \in L\}$ as the *query class* of (L, Υ).

Intuitively, a factorization of problem L partitions each instance x of L into a "data" part $D = \pi_1(x)$ and a "query" part $Q = \pi_2(x)$, and ρ is an inverse function that restores the original instance x from $\pi_1(x)$ and $\pi_2(x)$.

Example 4: Consider a decision problem L_s that is to determine, given a relation D, an attribute A of D and a constant c, whether there exists a tuple $t \in D$ such that $t[A] = c$. A factorization of this problem is (π_1, π_2, ρ), such that for each instance $x = (D, A, c)$ of the problem, $\pi_1(x)$ is the relation D, $\pi_2(x)$ is the pair (A, c), and $\rho(\pi_1(x), \pi_2(x))$ maps $\pi_1(x)$ and $\pi_2(x)$ back to x. With this factorization, its query class of L_s is $\mathcal{Q}_1$ given in Example 1, and its language of pairs is the same as S_1 given in Example 3.

As another example consider the BDS problem given in Example 2 and the factorization $\Upsilon_{\mathsf{BDS}} = (\pi_1, \pi_2, \rho)$ of its instances $x = (G, (u, v))$, where $\pi_1(x) = G$, $\pi_2(x) = (u, v)$, and ρ maps $\pi_1(x)$ and $\pi_2(x)$ back to x, as shown in Fig. 1. This yields a language $S_{(\mathsf{BDS},\Upsilon_{\mathsf{BDS}})}$ of pairs $\langle G, (u, v)\rangle$, where u is visited before v in the breadth-depth search. That is, we treat graph G as data and pair (u, v) as a query. □

We are now ready to formally define when decision problems can be made Π-tractable.

Definition 2: We say that a decision problem L can be *made* Π-*tractable* if there exists a factorization $\Upsilon = (\pi_1, \pi_2, \rho)$ of L such that the language $S_{(L,\Upsilon)}$ of pairs for (L, Υ) is Π-tractable, *i.e.,* the query class $L_{(Q,\Upsilon)}$ of (L, Υ) is Π-tractable. We use $\Pi\mathsf{T}_{\mathsf{P}}$ to denote the set of all decision problems that can be made Π-tractable. □

That is, L can be made Π-tractable if L can be factorized into data and query parts, such that after the data is preprocessed in PTIME, all the instances of L with the same data can be solved by evaluating their queries on the data in parallel polylog-time. There are possibly multiple factorizations for L. As long as *one of the factorizations* transforms instances of L to a Π-tractable query class and data sets, L can be made Π-tractable, since the factorization allows us to efficiently decide the membership of instances in L:

Proposition 1: *If a decision problem L can be made Π-tractable by means of a factorization $\Upsilon = (\pi_1, \pi_2, \rho)$, then $x \in L$ if and only if $\langle \pi_1(x), \pi_2(x)\rangle \in S_{L,\Upsilon}$.* □

Proof: If $x \in L$ then $\langle \pi_1(x), \pi_2(x)\rangle \in S_{(L,\Upsilon)}$ by the definition of $S_{(L,\Upsilon)}$. Conversely, if $\langle \pi_1(x), \pi_2(x)\rangle \in S_{(L,\Upsilon)}$ then there exists $x' \in L$ such that $\pi_1(x) = \pi_1(x')$ and $\pi_2(x) = \pi_2(x')$. Assume that $x \neq x'$. By the definition of factorizations, $x = \rho(\pi_1(x), \pi_2(x)) = \rho(\pi_1(x'), \pi_2(x')) = x'$, which contradicts our assumption. Thus $x = x'$ and $x \in L$. □

Example 5: Problem L_s described in Example 4 can be

made Π-tractable. Indeed, the factorization given there transforms L_s to the class $\mathcal{Q}_1$ of point-selection queries on relations, and as argued in Example 3, $\mathcal{Q}_1$ is Π-tractable.

Now recall BDS from Example 2. It is known that BDS is P-complete (cf. [21]), and is costly when graph G is big. We show that it is also in $\Pi\mathsf{T}_\mathsf{P}$. The factorization Υ_{BDS} given in Example 4 yields a language $S_{(\mathsf{BDS},\Upsilon_{\mathsf{BDS}})}$ of pairs. Given Υ_{BDS}, we define preprocessing $\Pi(\cdot)$ as the function that performs breadth-depth search on G based on the ordering on the vertices, and returns a list M consisting of all the nodes in V in the same order as they are visited during the search. Then $\Pi(G)$ is clearly in PTIME in $|G|$. Let S' be the language of pairs $\langle M,(u,v)\rangle$ such that u appears before v in M. Observe the following: (a) whether $(M,(u,v)) \in S'$ can be decided by binary searches on M, in $O(\log|M|)$ time; and (b) $\langle M,(u,v)\rangle \in S'$ if and only if $\langle G,(u,v)\rangle \in S_{(\mathsf{BDS},\Upsilon_{\mathsf{BDS}})}$. Hence by Definition 1, $S_{(\mathsf{BDS},\Upsilon_{\mathsf{BDS}})}$ is Π-tractable. Thus BDS can be made Π-tractable by Definition 2. □

Making query classes Π-tractable. Given a query class that is not Π-tractable, a natural question is whether we can make it Π-tractable by changing its data and query parts, similar to the case of decision problems? This is important since we want to know whether there exists an efficient transformation that makes our queries feasible on big data.

To answer this question, we treat query classes as decision problems and leverage Definition 2. We have seen that for each decision problem L and each factorization Υ of L, there exist a language $S_{(L,\Upsilon)}$ of pairs and a class $L_{(D,\Upsilon)}$ of queries associated with them. Conversely, for each class $\mathcal{Q}$ of queries, which is represented as a language $S_\mathcal{Q}$ of pairs $\langle D,Q\rangle$, there also exists a decision problem $L_\mathcal{Q} = \{D\#Q \mid \langle D,Q\rangle \in S_\mathcal{Q}\}$ associated with it, where # is a delimiter, and $L_\mathcal{Q} \subseteq \Sigma^*$. The problem $L_\mathcal{Q}$ is referred to as *the decision problem of* $\mathcal{Q}$, and can be stated as:

- Input: $D\#Q$.
- Question: Does $Q(D)$ evaluate to true?

Obviously, for any complexity class C, $S_\mathcal{Q} \in \mathsf{C}$ if and only if $L_\mathcal{Q} \in \mathsf{C}$. Moreover, a factorization $\Upsilon_{L_\mathcal{Q}}$ of $L_\mathcal{Q}$ can be readily defined that extracts Q and D from an instance $x = D\#Q$ of $L_\mathcal{Q}$. This yields the language $S_\mathcal{Q}$ of pairs as $S_{(L_\mathcal{Q},\Upsilon_{L_\mathcal{Q}})}$. In other words, $S_\mathcal{Q}$ *determines* a factorization of $L_\mathcal{Q}$.

One may explore various factorizations Υ for $L_\mathcal{Q}$, which may yield languages $S_{(L_\mathcal{Q},\Upsilon)}$ of pairs *different* from $S_\mathcal{Q}$. We refer to such an Υ as a *re-factorization* for $S_\mathcal{Q}$ and $\mathcal{Q}$.

Definition 3: We say that a class $\mathcal{Q}$ of queries *can be made* Π*-tractable* if the corresponding decision problem $L_\mathcal{Q}$ of $\mathcal{Q}$ can be made Π-tractable. We denote by $\Pi\mathsf{T}_\mathsf{Q}$ the set of all query classes that can be made Π-tractable. □

That is, $\mathcal{Q}$ is in $\Pi\mathsf{T}_\mathsf{Q}$ if it can be re-factorized by some factorization Υ to re-partition its data and query parts for preprocessing, such that $S_{(L_\mathcal{Q},\Upsilon)}$ is Π-tractable.

Figure 2 depicts the relationship among the three classes $\Pi\mathsf{T}^0_\mathsf{Q}$, $\Pi\mathsf{T}_\mathsf{P}$ and $\Pi\mathsf{T}_\mathsf{Q}$. From the definitions above we can see that the set $\Pi\mathsf{T}_\mathsf{Q}$ of all query classes that *can be made* Π-tractable corresponds to $\Pi\mathsf{T}_\mathsf{P}$. As will be seen in Section 7, this set *properly contains* the set $\Pi\mathsf{T}^0_\mathsf{Q}$ of all query classes that *are* Π-tractable unless $\mathsf{P} = \mathsf{NC}$. Moreover, $\leqslant^{\mathsf{NC}}_{fa}$ reductions (Section 5) are transformations for making a query class Π-tractable, and are compatible with the set $\Pi\mathsf{T}_\mathsf{Q}$.

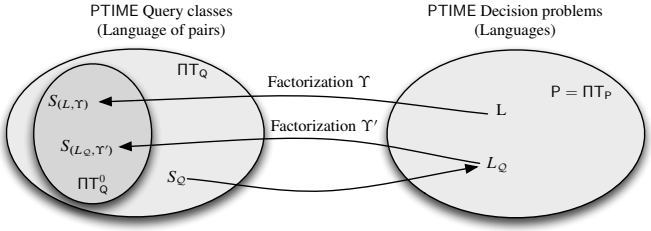

Figure 2: $\Pi\mathsf{T}^0_\mathsf{Q}$: Π-tractable query classes; $\Pi\mathsf{T}_\mathsf{P}$: decision problems that can be made Π-tractable; $\Pi\mathsf{T}_\mathsf{Q}$: query classes that can be made Π-tractable.

We use P (resp. NC) to denote the class of all decision problems that can be solved in PTIME (resp. NC) and the set of all PTIME (resp. NC) query classes interchangeably.

4. Π-Tractable Cases

Extending Examples 3 and 5, we next present more Π-tractable query classes, as well as decision problems that can be made tractable. We illustrate the effect of preprocessing and explore general strategies to make queries Π-tractable.

(1) Selection queries. We have seen from Example 1 that the class $\mathcal{Q}_1$ of point-selection queries is in $\Pi\mathsf{T}^0_\mathsf{Q}$, by using indices. This also extends to range-selection queries. Given a relation instance D of schema R, a range-selection (Boolean) query is to find whether there exists a tuple $t \in D$ such that $c_1 \leq t[A] \leq c_2$, where A is an attribute of R, and c_1 and c_2 are constants. As database texts tell us (*e.g.,* [34]), we may preprocess D by building B^+-trees. With these, such range queries on D can also be answered in $O(\log|D|)$ time.

(2) Searching in a list. Consider a decision problem L_1:

- Input: An unordered list M, and an element e.
- Question: Does e appear in M?

This problem is in $\Pi\mathsf{T}_\mathsf{P}$. Indeed, a factorization of L_1 is $\Upsilon_1 = (\pi_1,\pi_2,\rho)$ such that for any instance $x = (M,e)$ of L_1, $\pi_1(x) = M$ and $\pi_2(x) = e$, *i.e.,* it treats M as data and e as a query. The language $S_{(L_1,\Upsilon_1)}$ for (L_1,Υ_1) consists of pairs $\langle M,e\rangle$. One can define a preprocessing function $\Pi(\cdot)$ for $S_{(L_1,\Upsilon_1)}$ that sorts M into an ordered list, in $O(|M|\log|M|)$ time. After this, for all queries e, one can decide whether e is in M via binary search in $O(\log|M|)$ time.

(3) Minimum Range Queries (MRQ). Consider L_2 [18]:

- Input: An array $A[1,n]$ of n objects from a totally ordered domain, and i,j with $1 \leq i \leq j \leq n$.
- Question: Find $\mathsf{RMQ}_A(i,j) = \mathsf{arg\ min}_{i\leq k\leq j}\{A[k]\}$.

The range minimum query $\mathsf{RMQ}_A(i,j)$ returns the position of a minimum element in the sub-array $A[i,j]$. Obviously L_2 can be factorized into array $A[1,n]$ and query $\mathsf{RMQ}_A(i,j)$. As shown in [18], one may preprocess $A[1,n]$ by building an auxiliary structure of $O(n)$ bits, in PTIME, such that all range minimum queries $\mathsf{RMQ}_A(i,j)$ can be subsequently answered in $O(1)$ time by using the structure. Note that L_2 is a search problem, although it can be readily converted to a decision problem in $\Pi\mathsf{T}_\mathsf{P}$, *i.e.,* made Π-tractable.

(4) Lowest Common Ancestors (LCA). Consider L_3:

- Input: A DAG G, and nodes u and v in G.
- Question: Find $\mathsf{LCA}(u,v)$, *i.e.,* node w in G such that w is an ancestor of both u and v and moreover, w has no descendants that are also ancestors of u and v.

This is also a search problem. As shown in [5], G can be preprocessed by computing LCA for all pairs of nodes in G in $O(|G|^3)$ time. Then given any nodes (u, v) in G, $\mathsf{LCA}(u, v)$ can be found in $O(1)$ time; so its decision problem is in $\Pi\mathsf{T}_\mathsf{P}$.

As remarked earlier, it has been show in [39] that some natural classes of queries are in NC over ordered relational databases. These query classes are also in $\Pi\mathsf{T}^0_\mathsf{Q}$.

We next present three general strategies to put query classes $\mathcal{Q}$ in $\Pi\mathsf{T}^0_\mathsf{Q}$, not limited to any specific $\mathcal{Q}$.

(5) Query preserving compression. The key idea is as follows: for a class $\mathcal{Q}$ of queries, we preprocess a database D by finding a smaller database D_c via an efficient compression function, such that for *all* queries $Q \in \mathcal{Q}$, $Q(D) = Q(D_c)$, *i.e.,* the answer to Q in the smaller D_c. In contrast to lossless compression schemes (*e.g.,* [6, 9, 17]), query preserving compression is *relative to* $\mathcal{Q}$, *i.e.,* it generates small D_c that preserves the information *only relevant* to queries in $\mathcal{Q}$ rather than preserving the entire original D. Hence it often achieves a better compression ratio than lossless compression. Indeed, this approach has proven effective in answering graph queries on large social network graphs [16, 31, 32] and in cryptographic applications [24]. If the compression can be conducted in PTIME [16, 31, 32] and moreover, queries in $\mathcal{Q}$ can be answered in the compressed databases D_c in parallel polylog-time, perhaps by combining with other techniques such as indexing, then $\mathcal{Q}$ is Π-tractable, *i.e.,* $\mathcal{Q} \in \Pi\mathsf{T}^0_\mathsf{Q}$.

(6) Query answering using views. Another technique commonly used by database people is query answering using views [1, 23, 30]. Given a query $Q \in \mathcal{Q}$ and a set $\mathcal{V}$ of view definitions, query answering using views is to reformulate Q into another query Q' such that (a) Q and Q' are equivalent, *i.e.,* for all databases D, $Q(D) = Q'(D)$, and moreover, (b) Q' refers only to $\mathcal{V}$ and its extensions $\mathcal{V}(D)$.

When all queries in $\mathcal{Q}$ can be answered using views, $\mathcal{Q}$ is Π-tractable (in $\Pi\mathsf{T}^0_\mathsf{Q}$) if the following conditions are satisfied. (a) The views can be created and materialized in PTIME, as preprocessing. (b) The answer $Q(D)$ can be computed by evaluating Q' on the views $\mathcal{V}(D)$ in parallel polylog-time in $|Q|$ and $|D|$, where Q' is a reformulation of Q (*i.e.,* query rewriting; see the remark below Definition 1). This is done by possibly combining with other techniques for preprocessing. In practice $\mathcal{V}(D)$ is often much smaller than D. If $Q(D)$ can be found by using $\mathcal{V}(D)$ only, *without* accessing the original big D, then it is feasible to evaluate the queries on D.

(7) Incremental evaluation. Incremental techniques also allow us to effectively evaluate queries on big data [15, 37]. Given Q in a class $\mathcal{Q}$ of queries, as preprocessing we compute $Q(D)$ once. When D is updated by ΔD, instead of recomputing $Q(D \oplus \Delta D)$ starting from scratch, we incrementally compute ΔO such that $Q(D \oplus \Delta D) = Q(D) \oplus \Delta O$, to minimize unnecessary recomputation. In practice, ΔD is typically small; as a result, ΔO tends to be small as well, and is far more efficient to compute than to recompute $Q(D \oplus \Delta D)$. The benefit is particularly evident if there exists a bounded incremental algorithm for answering queries in $\mathcal{Q}$. As argued in [35], incremental algorithms should be analyzed in terms of $|\mathsf{CHANGED}| = |\Delta D| + |\Delta O|$, which represents the updating costs that are *inherent to* the incremental problem itself. An incremental algorithm is said to be *bounded* if its cost can be expressed as a function of $|\mathsf{CHANGED}|$, *i.e.,* it depends only on $|\mathsf{CHANGED}|$, *independent of* the original D. Then $\mathcal{Q}$ is Π-tractable (*i.e.,* $\mathcal{Q} \in \Pi\mathsf{T}^0_\mathsf{Q}$) if (a) the preprocessing $Q(D)$ is in PTIME, and (b) $Q(D \oplus \Delta D)$ can be incrementally computed in parallel polylog-time. If so, it is feasible to answer Q in response to changes to big data D.

As remarked in Section 1, we also use (bounded) incremental techniques to preprocess D in response ΔD.

Finally, we investigate two "hard" problems that, under certain conditions, can also be made Π-tractable (in $\Pi\mathsf{T}_\mathsf{P}$).

(8) Circuit Value Problem (CVP). CVP is stated as:

- Input: An encoding $\bar{\alpha}$ of a Boolean circuit α, inputs $x_1, \ldots, x_n$, and a designated output y.
- Question: Is output y of α true on inputs $x_1, \ldots, x_n$?

Informally, a Boolean circuit α is a DAG, in which a node can be (a) an input node x_i for $i \in [1, n]$, with indegree 0; (b) an output y, with outdegree 0, or (c) a gate denoting a Boolean operator, *e.g.,* $\wedge, \vee, \neg, \rightarrow$, which applies to its input (children) and feeds the result as an input for its parent nodes. It is a Boolean function that takes input $x_1, \ldots, x_n$, and returns the truth value of y. Its encoding $\bar{\alpha}$ is a sequence of tuples, one for each node in the DAG (see [21] for details).

CVP is perhaps the best known P-complete problem (cf. [21]). In Section 6 we will show that CVP can also be made Π-tractable, *i.e.,* CVP can be factorized to be feasible on big data (*i.e.,* when α is "big"). Nevertheless, we will show in Section 7 that there exist certain fixed factorizations of CVP that may not be evaluated on big data efficiently, *i.e.,* CVP is feasible on big data only after it is properly factorized.

(9) Vertex Cover (VC). VC is stated as follows [20]:

- Input: Graph $G = (V, E)$ and positive integer $K \leq |V|$.
- Question: Is there a vertex cover of size $\leq K$ for G?

A vertex cover of G is a set $V' \subseteq V$ such that for each edge $(u, v) \in E$, at least one of u and v is in V'.

It is known that VC is NP-complete in general (cf. [20]). However, as shown by the study of parameterized complexity theory [19], its instances can be preprocessed by Buss' Kernelization in $O(|E|)$ time, such that when K is fixed, it is in $O(1)$ time to decide whether there exists a vertex cover of size K or less. That is, *when K is fixed*, VC is in $\Pi\mathsf{T}_\mathsf{P}$.

5. NC Reducibility with Factorizations

There are several important and practical questions in connection with Π-tractability. Given a class $\mathcal{Q}$ of queries, if $\mathcal{Q}$ is not Π-tractable, can we make it Π-tractable by means of efficient transformations? If so, what kind of transformations should we use? Furthermore, how large is $\Pi\mathsf{T}_\mathsf{Q}$, the set of all query classes that can be made Π-tractable? Can we determine whether a decision problem can be made Π-tractable by "reducing" it to another problem in $\Pi\mathsf{T}_\mathsf{P}$? Moreover, is there a natural problem that is complete for $\Pi\mathsf{T}_\mathsf{P}$, *i.e.,* the "hardest" problem in $\Pi\mathsf{T}_\mathsf{P}$?

We answer these questions in the next two sections. In this section we introduce a form of reductions $\leqslant^{\mathsf{NC}}_{fa}$ to specify transformations from one problem to another, and to make a query class Π-tractable. In the next section we will use $\leqslant^{\mathsf{NC}}_{fa}$ to relate a problem to the entire complexity class $\Pi\mathsf{T}_\mathsf{P}$.

Below we first introduce $\leqslant^{\mathsf{NC}}_{fa}$ for decision problems (Section 5.1). We then extend $\leqslant^{\mathsf{NC}}_{fa}$ to query classes (Section 5.2). We show that $\leqslant^{\mathsf{NC}}_{fa}$ reductions are transitive and are compatible with $\Pi\mathsf{T}_\mathsf{P}$. As will be seen shortly, these properties are

not as trivial as their counterparts for PTIME reductions.

5.1 Reducibility for Decision Problems

We start with a form of many-one reductions for decision problems in $\Pi\mathsf{T}_\mathsf{P}$, which allow many distinct instances of one problem to be mapped to a single instance of another.

In contrast to PTIME reductions that we use to prove, *e.g.,* that a problem is NP-complete, reductions for $\Pi\mathsf{T}_\mathsf{P}$ are more intriguing. They are defined in terms of factorizations, and involve both the data and query parts determined by the factorizations. In addition, such reductions have to be *compatible* with $\Pi\mathsf{T}_\mathsf{P}$: if problem A is reducible to another problem B and if B can be made Π-tractable, then so must be A. In light of this, PTIME reductions no longer work here. Furthermore, like our familiar PTIME reductions, such reductions should be transitive: if A can be reduced to B and B can be reduced to C, then A is also reducible to C.

Taking these into account, we now define $\leqslant_{fa}^{\mathsf{NC}}$ reductions.

Definition 4: A problem (language) L_1 is said to be NC-*factor reducible* to another problem (language) L_2, denoted by $L_1 \leqslant_{fa}^{\mathsf{NC}} L_2$, if there exist factorizations $\Upsilon_1 = (\pi_1^1, \pi_2^1, \rho_1)$ and $\Upsilon_2 = (\pi_1^2, \pi_2^2, \rho_2)$ of L_1 and L_2, respectively, and NC functions $\alpha(\cdot)$ and $\beta(\cdot)$, such that for all D and Q in Σ^*, $\langle D, Q\rangle \in S_{(L_1,\Upsilon_1)}$ if and only if $\langle \alpha(D), \beta(Q)\rangle \in S_{(L_2,\Upsilon_2)}$. □

In contrast to PTIME reductions, a NC-factor reduction is defined in terms of (a) two NC functions $\alpha(\cdot)$ and $\beta(\cdot)$ rather than a single PTIME function, and (b) factorizations: $L_1 \leqslant_{fa}^{\mathsf{NC}} L_2$ if *one of* the factorizations Υ_1 of L_1 can be reduced to *one of* the factorizations Υ_2 of L_2, such that $\alpha(\cdot)$ and $\beta(\cdot)$ transform the data and query parts of (L_1, Υ_1) to their counterparts of (L_2, Υ_2), respectively.

We now verify that $\leqslant_{fa}^{\mathsf{NC}}$ has the necessary properties of reductions. We first show that $\leqslant_{fa}^{\mathsf{NC}}$ is transitive.

Lemma 2: *The reducibility relation* $\leqslant_{fa}^{\mathsf{NC}}$ *is transitive,* i.e., *if* $L_1 \leqslant_{fa}^{\mathsf{NC}} L_2$ *and* $L_2 \leqslant_{fa}^{\mathsf{NC}} L_3$, *then also* $L_1 \leqslant_{fa}^{\mathsf{NC}} L_3$. □

Proof: Given that $L_1 \leqslant_{fa}^{\mathsf{NC}} L_2$ and $L_2 \leqslant_{fa}^{\mathsf{NC}} L_3$, we have the following factorizations by the definition of $\leqslant_{fa}^{\mathsf{NC}}$:

- For L_1: $\Upsilon_1 = (\pi_1^1, \pi_2^1, \rho_1)$;
- For L_2: $\Upsilon_2 = (\pi_1^2, \pi_2^2, \rho_2)$ and $\Upsilon_2' = (\sigma_1^2, \sigma_2^2, \rho_2')$; and
- For L_3: $\Upsilon_3 = (\pi_1^3, \pi_2^3, \rho_3)$.

In addition, there exist NC functions $\alpha_1(\cdot)$, $\alpha_2(\cdot)$, $\beta_1(\cdot)$ and $\beta_2(\cdot)$ such that for all $D_1, Q_1, D_2, Q_2 \in \Sigma^*$:

$$\begin{array}{ll} \langle D_1, Q_1\rangle \in S_{(L_1,\Upsilon_1)} & \text{iff } \langle \alpha_1(D_1), \beta_1(Q_1)\rangle \in S_{(L_2,\Upsilon_2)}, \\ \langle D_2, Q_2\rangle \in S_{(L_2,\Upsilon_2')} & \text{iff } \langle \alpha_2(D_2), \beta_2(Q_2)\rangle \in S_{(L_3,\Upsilon_3)}. \end{array}$$

To show that $L_1 \leqslant_{fa}^{\mathsf{NC}} L_3$, we have to define factorizations of L_1 and L_3 and NC functions $\alpha(\cdot)$ and $\beta(\cdot)$ as required by the definition of $\leqslant_{fa}^{\mathsf{NC}}$. We cannot simply compose (α_1, β_1) and (α_2, β_2) to a reduction $(\alpha_2 \circ \alpha_1, \beta_2 \circ \beta_1)$ as in the case for a standard proof of transitivity of reductions, because α_2 and β_2 may rely on both the query and data parts resulting from the reduction (α_1, β_1). The solution lies in padding both the query and data parts as follows. We start by defining a factorization $\Upsilon_1' = (\sigma_1^1, \sigma_2^1, \rho_1')$ of L_1. For all $x \in L_1$, we define $\sigma_1^1(x) = \pi_1^1(x)@\pi_2^1(x)$, where @ is a special symbol that is not used anywhere else. In addition, we let $\sigma_2^1(x) = \sigma_1^1(x)$, and define $\rho_1'(x_1@x_2, x_1@x_2) = \rho_1(x_1, x_2)$. More specifically, consider $S_{(L_1,\Upsilon_1')}$, $S_{(L_3,\Upsilon_3)}$ and the following functions: for any $D_1, Q_1 \in \Sigma^*$,

$$\begin{array}{l} \alpha(D_1) = \alpha_2(\sigma_1^2(\rho_2(\alpha_1'(D_1), \beta_1'(D_1)))), \\ \beta(Q_1) = \beta_2(\sigma_2^2(\rho_2(\alpha_1'(Q_1), \beta_1'(Q_1)))), \end{array}$$

where $\alpha_1'(D_1) = \alpha_1(r)$ if $D_1 = r@s$ and $\beta_1'(D_1) = \beta_1(s)$ if $D_1 = r@s$. Similarly for $\alpha_1'(Q_1)$ and $\beta_1'(Q_1)$. Given these, we verify that for any $D_1, Q_1 \in \Sigma^*$,

$$\langle D_1, Q_1\rangle \in S_{(L_1,\Upsilon_1')} \text{ iff } \langle \alpha(D_1), \beta(Q_1)\rangle \in S_{(L_3,\Upsilon_3)}.$$

Indeed, one can readily verify the following:

$$\begin{array}{l} \langle D_1, Q_1\rangle \in S_{(L_1,\Upsilon_1')} \text{ iff } D_1 = \sigma_1^1(x), Q_1 = \sigma_2^1(x) \text{ for some } x \in L_1 \\ \text{iff } D_1 = Q_1 = \pi_1^1(x)@\pi_2^1(x) \\ \text{iff } D_1 = Q_1 = D_1'@Q_1', \langle D_1', Q_1'\rangle \in S_{(L_1,\Upsilon_1)} \\ \text{iff } \langle \alpha_1'(D_1), \beta_1'(Q_1)\rangle \in S_{(L_2,\Upsilon_2)} \\ \text{iff } \alpha_1'(D_1) = \pi_1^2(y), \beta_1'(Q_1) = \pi_2^2(y) \text{ for some } y \in L_2 \\ \text{iff } \rho_2(\alpha_1'(D_1), \beta_1'(Q_1)) \in L_2 \\ \text{iff } \langle \sigma_1^2(\rho_2(\alpha_1'(D_1), \beta_1'(Q_1))), \sigma_2^2(\rho_2(\alpha_1'(D_1), \beta_1'(Q_1)))\rangle \in S_{(L_2,\Upsilon_2)} \\ \text{iff } \langle \sigma_1^2(\rho_2(\alpha_1'(D_1), \beta_1'(D_1))), \sigma_2^2(\rho_2(\alpha_1'(Q_1), \beta_1'(Q_1)))\rangle \in S_{(L_2,\Upsilon_2)} \\ \text{iff } \langle \alpha_2(\sigma_1^2(\rho_2(\alpha_1'(D_1), \beta_1'(D_1)))), \beta_2(\sigma_2^2(\rho_2(\alpha_1'(Q_1), \beta_1'(Q_1))))\rangle \\ \qquad \in S_{(L_3,\Upsilon_3)} \\ \text{iff } \langle \alpha(D_1), \beta(Q_1)\rangle \in S_{(L_3,\Upsilon_3)}, \end{array}$$

as desired. Hence, $L_1 \leqslant_{fa}^{\mathsf{NC}} L_3$. □

We next verify that $\leqslant_{fa}^{\mathsf{NC}}$ is *compatible with* $\Pi\mathsf{T}_\mathsf{P}$.

Lemma 3: *For all problems* L_1 *and* L_2, *if* $L_1 \leqslant_{fa}^{\mathsf{NC}} L_2$ *and* L_2 *is in* $\Pi\mathsf{T}_\mathsf{P}$, *then* L_1 *is also in* $\Pi\mathsf{T}_\mathsf{P}$. *That is, if* $L_1 \leqslant_{fa}^{\mathsf{NC}} L_2$ *and* L_2 *can be made* Π*-tractable, then so can be* L_1. □

Lemma 3 tells us how to show that a problem L_1 can be made Π-tractable, *i.e.,* L_1 is in $\Pi\mathsf{T}_\mathsf{P}$: pick a problem $L_2 \in \Pi\mathsf{T}_\mathsf{P}$ and show that $L_1 \leqslant_{fa}^{\mathsf{NC}} L_2$. It also tells us how to show that a problem L_2 cannot be made Π-tractable: if $L_1 \leqslant_{fa}^{\mathsf{NC}} L_2$ and $L_1 \notin \Pi\mathsf{T}_\mathsf{P}$ then $L_2 \notin \Pi\mathsf{T}_\mathsf{P}$.

Proof: From $L_1 \leqslant_{fa}^{\mathsf{NC}} L_2$ we know that there exist factorizations $\Upsilon_1 = (\pi_1^1, \pi_2^1, \rho_1)$, $\Upsilon_2 = (\pi_1^2, \pi_2^2, \rho_2)$ and NC functions $\alpha(\cdot)$ and $\beta(\cdot)$ such that for all $D_1, Q_1 \in \Sigma^*$,

$$\langle D_1, Q_1\rangle \in S_{(L_1,\Upsilon_1)} \text{ iff } \langle \alpha(D_1), \beta(Q_1)\rangle \in S_{(L_2,\Upsilon_2)}.$$

Furthermore, since L_2 is in $\Pi\mathsf{T}_\mathsf{P}$, by Definitions 1 and 2, there must exist a factorization $\Upsilon_2' = (\sigma_1^2, \sigma_2^2, \rho_2')$ of L_2, a PTIME preprocessing function $\Pi(\cdot)$, and a language S_2' of pairs in NC, such that for all $D_2, Q_2 \in \Sigma^*$,

$$\langle D_2, Q_2\rangle \in S_{(L_2,\Upsilon_2')} \text{ iff } \langle \Pi(D_2), Q_2\rangle \in S_2'.$$

To show that L_1 can be made Π-tractable, we first argue that L_1 is also NC-factor reducible to L_2 with the factorization Υ_2' of L_2 rather than Υ_2. Indeed, we can construct an NC-factor reduction that meets this requirement along the same lines as the proof of Lemma 2, by changing the factorization Υ_1 of L_1. This yields a factorization $\Upsilon_1' = (\sigma_1^1, \sigma_2^1, \rho_1')$ of L_1 and NC functions $\alpha'(\cdot)$ and $\beta'(\cdot)$ such that

$$\begin{array}{ll} \langle D_1, Q_1\rangle \in S_{(L_1,\Upsilon_1')} & \text{iff } \langle \alpha'(D_1), \beta'(Q_1)\rangle \in S_{(L_2,\Upsilon_2')} \\ & \text{iff } \langle \Pi(\alpha'(D_1)), \beta'(Q_1)\rangle \in S_2'. \end{array}$$

Define a language S_2'' of pairs such that $\langle a, b\rangle \in S_2''$ if and only if $\langle a, \beta'(b)\rangle \in S_2'$. Obviously, if S_2' is in NC then so is S_2''. Define $\Pi'(D_1) = \Pi(\alpha'(D_1))$. Then we have that

$$\langle D_1, Q_1\rangle \in S_{(L_1,\Upsilon_1')} \text{ iff } \langle \Pi'(D_1), Q_1\rangle \in S_2''.$$

Note that $\Pi'(\cdot)$ is in PTIME since $\Pi(\cdot)$ is in PTIME, $\alpha'(\cdot)$ is in NC, and $\mathsf{NC} \subseteq \mathsf{P}$. Moreover, S_2'' is in NC since S_2' is in NC by the assumption above. Therefore, L_1 is also in $\Pi\mathsf{T}_\mathsf{P}$. □

5.2 Reducibility for Query Classes

In view of the relationship between query classes $\mathcal{Q}$ and decision problem $L_{\mathcal{Q}}$, as we have seen in Section 3, we can now readily define $\leqslant_{fa}^{\mathsf{NC}}$ for query classes as follows.

Definition 5: For classes $\mathcal{Q}_1$ and $\mathcal{Q}_2$ of queries, we say that $\mathcal{Q}_1$ is NC-*factor reducible* to $\mathcal{Q}_2$, also denoted by $\mathcal{Q}_1 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_2$, if $L_{\mathcal{Q}_1}$ is NC-*factor reducible* to $L_{\mathcal{Q}_2}$, where $L_{\mathcal{Q}_1}$ and $L_{\mathcal{Q}_2}$ are the decision problems of $\mathcal{Q}_1$ and $\mathcal{Q}_2$, respectively. □

That is, $\mathcal{Q}_1 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_2$ if $L_{\mathcal{Q}_1} \leqslant_{fa}^{\mathsf{NC}} L_{\mathcal{Q}_2}$, *i.e., there exist* factorization Υ_1 of the decision problem $L_{\mathcal{Q}_1}$ of $\mathcal{Q}_1$ and factorization Υ_2 of the decision problem $L_{\mathcal{Q}_2}$ of $\mathcal{Q}_2$, such that $S_{(L_{\mathcal{Q}_1},\Upsilon_1)}$ can be transformed to $S_{(L_{\mathcal{Q}_2},\Upsilon_2)}$ in NC (recall the decision problem for a query class from Section 3).

As an immediate result of Lemmas 2 and 3, we have:

Corollary 4: *For all classes $\mathcal{Q}_1$, $\mathcal{Q}_2$ and $\mathcal{Q}_3$ of queries,*

(1) if $\mathcal{Q}_1 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_2$ and $\mathcal{Q}_2 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_3$, then $\mathcal{Q}_1 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_3$; and

(2) if $\mathcal{Q}_1 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_2$ and $\mathcal{Q}_2$ is in $\Pi\mathsf{T}_{\mathsf{Q}}$, then $\mathcal{Q}_1$ is also in $\Pi\mathsf{T}_{\mathsf{Q}}$. That is, if $\mathcal{Q}_1 \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}_2$ and $\mathcal{Q}_2$ can be made Π-tractable, then $\mathcal{Q}_1$ can also be made Π-tractable. □

In other words, $\leqslant_{fa}^{\mathsf{NC}}$ is compatible with $\Pi\mathsf{T}_{\mathsf{Q}}$, the set of query classes that *can be made* Π-tractable. In contrast, $\leqslant_{fa}^{\mathsf{NC}}$ is not compatible with $\Pi\mathsf{T}_{\mathsf{Q}}^0$, the set of query classes that *are* Π-tractable. Indeed, as will be seen in Section 6, there exists a query class for the Circuit Value Problem (CVP) that is not Π-tractable, but it is NC-factor reducible to a Π-tractable class of queries, *i.e.,* it can be made Π-tractable.

In Section 7 we will give a notion of F-reductions, which preserve the factorizations $S_{\mathcal{Q}_1}$ and $S_{\mathcal{Q}_2}$ for $L_{\mathcal{Q}_1}$ and $L_{\mathcal{Q}_2}$, *i.e.,* they do not allow re-factorizations of $\mathcal{Q}_1$ and $\mathcal{Q}_2$. As will be seen there, F-reductions are compatible with $\Pi\mathsf{T}_{\mathsf{Q}}^0$.

6. A Complete Problem for $\Pi\mathsf{T}_{\mathsf{P}}$

We next answer the question whether there exists a natural problem that is complete for $\Pi\mathsf{T}_{\mathsf{P}}$. A complete problem for $\Pi\mathsf{T}_{\mathsf{P}}$ is one that is no easier to solve than any other problem in the complexity class $\Pi\mathsf{T}_{\mathsf{P}}$ of all problems that can be made Π-tractable. That is, they capture the difficulty intrinsic of the $\Pi\mathsf{T}_{\mathsf{P}}$ class. Similarly, we answer this question for $\Pi\mathsf{T}_{\mathsf{Q}}$, *i.e.,* for query classes that can be made Π-tractable.

The main result of this section is that $\Pi\mathsf{T}_{\mathsf{P}}$ has complete problems. This is not evident, since there are complexity classes for which no complete problem exists. As a consequence of this result, it follows that all problems in P and all query classes that are in PTIME can be made Π-tractable.

We start by defining $\Pi\mathsf{T}_{\mathsf{P}}$-complete problems. Recall our familiar notion of NP-complete problems, *e.g.,* the $3\mathsf{SAT}$ problem and the Vertex Cover problem (VC) [33]. Such problems are in the complexity class NP, and moreover, all problems in NP can be reduced to them in PTIME. Along the same lines, we define $\Pi\mathsf{T}_{\mathsf{P}}$-complete problems.

Definition 6: A problem L is $\Pi\mathsf{T}_{\mathsf{P}}$-*hard* under NC-factor reducibility if $L' \leqslant_{fa}^{\mathsf{NC}} L$ for all decision problems L' in $\Pi\mathsf{T}_{\mathsf{P}}$. A problem L is $\Pi\mathsf{T}_{\mathsf{P}}$-*complete* under NC-factor reducibility if L can be made Π-tractable itself and L is $\Pi\mathsf{T}_{\mathsf{P}}$-hard.

A class $\mathcal{Q}$ of queries is said to be $\Pi\mathsf{T}_{\mathsf{Q}}$-*hard under* NC-*factor reducibility* if $\mathcal{Q}' \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}$ for all query classes $\mathcal{Q}'$ in $\Pi\mathsf{T}_{\mathsf{Q}}$. It is $\Pi\mathsf{T}_{\mathsf{Q}}$-*complete under* NC-*factor reducibility* if it can be made Π-tractable and is $\Pi\mathsf{T}_{\mathsf{Q}}$-hard. □

The need for studying these complete problems is evident. If there exists a $\Pi\mathsf{T}_{\mathsf{P}}$-complete problem L under $\leqslant_{fa}^{\mathsf{NC}}$, then given a problem L', we can determine whether $L' \in \Pi\mathsf{T}_{\mathsf{P}}$ by checking whether $L' \leqslant_{fa}^{\mathsf{NC}} L$. Similarly, if we know that a query class $\mathcal{Q}$ is $\Pi\mathsf{T}_{\mathsf{Q}}$-complete under $\leqslant_{fa}^{\mathsf{NC}}$, then to decide whether a query class $\mathcal{Q}'$ can be *made* Π-tractable, we simply check whether $\mathcal{Q}' \leqslant_{fa}^{\mathsf{NC}} \mathcal{Q}$. Further, as will be seen shortly, such a complete query class tells us how large $\Pi\mathsf{T}_{\mathsf{Q}}$ is.

Complete problems and complete query classes. No matter how important, not every complexity class has a complete problem. Indeed, while a number of complete problems have been established for NP [20] and for P [21], it is known that polylog-space has no complete problems under log-space reductions (cf. [26]), for instance.

The good news is that there exist a $\Pi\mathsf{T}_{\mathsf{P}}$-complete problem and a $\Pi\mathsf{T}_{\mathsf{Q}}$-complete query class under $\leqslant_{fa}^{\mathsf{NC}}$. To see this, recall the Breadth-Depth Search problem (BDS), its factorization Υ_{BDS} and language $S_{(\mathsf{BDS},\Upsilon_{\mathsf{BDS}})}$ of pairs presented in Examples 4 and 5. We use $\mathcal{Q}_{\mathsf{BDS}}$ to denote the query language $\mathcal{Q}_{(\mathsf{BDS},\Upsilon_{\mathsf{BDS}})}$ represented by $S_{(\mathsf{BDS},\Upsilon_{\mathsf{BDS}})}$.

Theorem 5: *Under NC-factor reducibility,*

(1) BDS is $\Pi\mathsf{T}_{\mathsf{P}}$-complete; and

(2) $\mathcal{Q}_{\mathsf{BDS}}$ is $\Pi\mathsf{T}_{\mathsf{Q}}$-complete. □

Proof: We show that BDS is $\Pi\mathsf{T}_{\mathsf{P}}$-complete. The proof for the $\Pi\mathsf{T}_{\mathsf{Q}}$-completeness of $\mathcal{Q}_{\mathsf{BDS}}$ under $\leqslant_{fa}^{\mathsf{NC}}$ is similar.

We have already shown that BDS can be made Π-tractable and is thus in $\Pi\mathsf{T}_{\mathsf{P}}$, in Example 5. We next show that BDS is $\Pi\mathsf{T}_{\mathsf{P}}$-hard under $\leqslant_{fa}^{\mathsf{NC}}$. That is, for all problems L that can be made Π-tractable, $L \leqslant_{fa}^{\mathsf{NC}}$ BDS. Observe the following:

(a) If L can be made Π-tractable, then $L \in \mathsf{P}$.

(b) BDS is P-complete (cf. [21]).

To see (a), note that the solution to any instance of L can be computed in PTIME, by combining the preprocessing step (PTIME) and query evaluation (NC), since $\mathsf{NC} \subseteq \mathsf{P}$.

Based on these, we show that $L \leqslant_{fa}^{\mathsf{NC}}$ BDS. Consider a factorization $\Upsilon_L = (\pi_1, \pi_2, \rho)$ that, for any instance x of L, defines $\pi_1(x) = \pi_2(x) = x$ and $\rho(x, x) = x$. Since BDS is P-complete and L is in P, this tells us that there exists an NC function $h(\cdot)$ such that for any $x \in \Sigma^*$, $x \in L$ if and only if $h(x)$ is in BDS. Then there exist NC functions $\alpha(\cdot)$ and $\beta(\cdot)$ such that $\alpha(x)$ corresponds to the undirected graph G in a BDS instance with a numbering on its vertices, as part of $h(x)$, and $\beta(x)$ corresponds to a pair of nodes (u, v) in G as the other part of $h(x)$. Hence, for all $\langle x, x\rangle \in S_{(L,\Upsilon_L)}$,

$$\langle x, x\rangle \in S_{(L,\Upsilon_L)} \text{ iff } \langle \alpha(x), \beta(x)\rangle \in S_{(\mathsf{BDS},\Upsilon_{\mathsf{BDS}})},$$

where $S_{(\mathsf{BDS},\Upsilon_{\mathsf{BDS}})}$ is the language of pairs for BDS and the factorization Υ_{BDS} given in Section 4. Hence, $L \leqslant_{fa}^{\mathsf{NC}}$ BDS by Definition 4. Thus BDS is $\Pi\mathsf{T}_{\mathsf{P}}$-hard under $\leqslant_{fa}^{\mathsf{NC}}$. □

The proof above also tells us the following.

Corollary 6: *(1) All problems in P can be made Π-tractable. (2) All query classes that are in PTIME can be made Π-tractable via NC-factor reductions.* □

That is, the set $\Pi\mathsf{T}_{\mathsf{P}}$ of all decision problems that can be made Π-tractable is large enough to cover all problems in P. Hence $\mathsf{P} = \Pi\mathsf{T}_{\mathsf{P}}$ since all problems that can be made Π-tractable are also in P. In particular, the Circuit Value Problem (CVP) discussed in Section 5 can be made Π-tractable, *i.e.,* there exists a factorization of CVP that allows instances

of CVP to be transformed to instances of BDS, such that after preprocessing the input circuits of CVP instances, those instances can be solved in parallel polylog-time.

Furthermore, all PTIME query classes can be made Π-tractable. These include query classes in $\Pi\mathsf{T}^0_\mathsf{Q}$ and beyond. Indeed, as will be seen in Section 7, there exists a factorization Υ_0 of CVP such that the query class $\mathsf{CVP}_{(Q,\Upsilon_0)}$ of (CVP, Υ_0) is *not* Π-tractable unless $\mathsf{P} = \mathsf{NC}$. That is, by preprocessing only the data part given in the particular factorization Υ_0, CVP instances may not be solvable in parallel polylog-time. Nevertheless, $\mathsf{CVP}_{(Q,\Upsilon_0)} \leqslant^{\mathsf{NC}}_{fa} \mathcal{Q}_{\mathsf{BDS}}$, *i.e.,* it can be made Π-tractable via re-factorizations. In general, a class $\mathcal{Q}$ of queries that are in PTIME may *not* necessarily be Π-tractable, but it can be made Π-tractable via re-factorizations as assured by Corollary 6.

To compare NC and $\Pi\mathsf{T}_\mathsf{P}$, observe that $\mathsf{NC} \subseteq \Pi\mathsf{T}_\mathsf{P}$ by Definition 2 for problems that can be made Π-tractable. Nevertheless, we do not know whether $\mathsf{NC} = \Pi\mathsf{T}_\mathsf{P}$. Indeed, this question is equivalent to the open problem whether $\mathsf{NC} = \mathsf{P}$.

Problems that cannot be made Π-tractable. At this point, a natural question is what problems are *not* in $\Pi\mathsf{T}_\mathsf{P}$. Theorem 5 tells us that unless $\mathsf{P} = \mathsf{NP}$, none of NP-complete problems is in $\Pi\mathsf{T}_\mathsf{P}$. In other words, problems such as the 3SAT problem and VC mentioned earlier cannot be made Π-tractable, no matter what factorizations are considered.

Corollary 7: *Unless* $\mathsf{P} = \mathsf{NP}$*, no* NP*-complete problems are* Π*-tractable.* □

Proof: Assume by contradiction that there exists an NP-complete problem that can be made Π-tractable. Then by Theorem 5, $L \leqslant^{\mathsf{NC}}_{fa}$ BDS by NC reductions. Nevertheless, BDS is in P. Hence this would lead to $\mathsf{P} = \mathsf{NP}$. □

7. Factorization Preserving Reductions

The NC-factor reductions allow us to transform one query class $\mathcal{Q}_1$ to another query class $\mathcal{Q}_2$ in $\Pi\mathsf{T}_\mathsf{Q}$ by means of re-factorizations. More specifically, suppose that $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are represented by languages S_1 and S_2 of pairs $\langle D_1, Q_1\rangle$ and $\langle D_2, Q_2\rangle$, respectively, and that $\mathcal{Q}_1 \leqslant^{\mathsf{NC}}_{fa} \mathcal{Q}_2$. Then there exist factorizations Υ_1 and Υ_2 of S_1 and S_2, respectively (see Sections 3 and 5), which yield languages $S_{(\mathcal{Q}_1,\Upsilon_1)}$ and $S_{(\mathcal{Q}_2,\Upsilon_2)}$ of pairs $\langle D'_1, Q'_1\rangle$ and $\langle D'_2, Q'_2\rangle$, such that D'_1 and Q'_1 can be transformed to D'_2 and Q'_2 via NC functions, respectively. Observe that Q'_1 may *no longer* be a query in the class $\mathcal{Q}_1$; similarly for Q'_2. That is, such reductions *re-factorize* S_1 and S_2, in which the original query classes $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are *not* preserved. In other words, $\mathcal{Q}_1$ is *made* Π-tractable by reducing it to $\mathcal{Q}_2$ with such re-factorizations.

This raises a natural question: what happens if we require to transform the query part Q_1 of S_1 to Q_2 of S_2, and the data part D_1 of S_1 to D_2 of S_2, preserving the original query classes $\mathcal{Q}_1$ and $\mathcal{Q}_2$? That is, one may want to use a form of reductions that preserve *fixed* query classes. The practical need for this emerges when we want to study, *e.g.,* query rewriting from $\mathcal{Q}_1$ to a fixed language $\mathcal{Q}_2$. This section studies such reductions, referred to as *F-reductions*. We define F-reductions and study their properties in Section 7.1.

We show that unless $\mathsf{P} = \mathsf{NC}$, there exists a class of PTIME queries that is not Π-tractable and moreover, it is not F-reducible to any class of Π-tractable queries, in Section 7.2. This tells us that without proper factorizations, it may not be feasible to evaluate PTIME queries on big data, and further highlights the need for appropriate preprocessing. This result separates $\Pi\mathsf{T}^0_\mathsf{Q}$ from the set P of all PTIME query classes unless $\mathsf{P} = \mathsf{NC}$. We also show that under F-reductions, it is rather hard to find a query class complete for $\Pi\mathsf{T}^0_\mathsf{Q}$, in Section 7.3. Indeed, the existence of such a class is closely related to the big open question: whether $\mathsf{P} = \mathsf{NC}$.

7.1 F-Reducibility

We define F-reductions on language of pairs and on query classes, rather than on decision problems, as follows.

Definition 7: A language S_1 of pairs is *F-reducible* to another language S_2 of pairs, denoted by $S_1 \leqslant^{\mathsf{NC}}_F S_2$, if there exist NC functions $\alpha(\cdot)$ and $\beta(\cdot)$ such that for all D and Q in Σ^*, $\langle D, Q\rangle \in S_1$ if and only if $\langle \alpha(D), \beta(Q)\rangle \in S_2$.

A class $\mathcal{Q}_1$ of queries is *F-reducible* to another query class $\mathcal{Q}_2$, denoted by $\mathcal{Q}_1 \leqslant^{\mathsf{NC}}_F \mathcal{Q}_2$, if $S_1 \leqslant^{\mathsf{NC}}_F S_2$, where S_1 and S_2 are the languages of pairs for $\mathcal{Q}_1$ and $\mathcal{Q}_2$, respectively. □

Compared to NC-factor reductions given in Definition 4, a F-reduction does not allow re-factorizations of S_1 and S_2. In terms of decision problems L_1 and L_2, F-reductions are defined on fixed factorizations Υ_1 of L_1 and Υ_2 of L_2. These reductions preserve the factorizations and their corresponding languages $S_{(L_1,\Upsilon_1)}$ and $S_{(L_2,\Upsilon_2)}$ in the transformations, and are used to study the relative difficulty of the data and query parts in $S_{(L_1,\Upsilon_1)}$ with their counterparts in $S_{(L_2,\Upsilon_2)}$.

Along the same lines as Lemmas 2 and 3, we show that F-reductions are transitive and compatible with $\Pi\mathsf{T}^0_\mathsf{Q}$. In contrast to $\leqslant^{\mathsf{NC}}_{fa}$ that is compatible with $\Pi\mathsf{T}_\mathsf{Q}$, the set of all query classes that can be *made* Π-tractable, $\leqslant^{\mathsf{NC}}_F$ is compatible with $\Pi\mathsf{T}^0_\mathsf{Q}$, the set of all query classes that *are* Π-tractable.

Lemma 8: *For any languages* S_1, S_2 *and* S_3 *of pairs,*

- *if* $S_1 \leqslant^{\mathsf{NC}}_F S_2$ *and* $S_2 \leqslant^{\mathsf{NC}}_F S_3$*, then also* $S_1 \leqslant^{\mathsf{NC}}_F S_3$*; and*
- *if* $S_1 \leqslant^{\mathsf{NC}}_F S_2$ *and* S_2 *is in* $\Pi\mathsf{T}^0_\mathsf{Q}$*, then* S_1 *is also in* $\Pi\mathsf{T}^0_\mathsf{Q}$*. That is,* $\leqslant^{\mathsf{NC}}_F$ *is compatible with* $\Pi\mathsf{T}^0_\mathsf{Q}$*.* □

7.2 Separation: PTIME and Π-Tractable Queries

Corollary 6 tells us that under NC-factor reductions, all PTIME query classes can be made Π-tractable by reduction to a query class that can be made Π-tractable. In contrast, when it comes to F-reductions, this is *no longer* possible.

Theorem 9: *Unless* $\mathsf{P} = \mathsf{NC}$*, there exists a query class that is in* P*, but it is not* Π*-tractable and moreover, it is not F-reducible to any* Π*-tractable class of queries.* □

Theorem 9 distinguishes the set $\Pi\mathsf{T}^0_\mathsf{Q}$ of all Π-tractable query classes from the set P of all PTIME query classes. From this and Corollary 6, we can see the connection and differences between $\Pi\mathsf{T}^0_\mathsf{Q}$ and P as follows.

(1) While a Π-tractable query class must be in P, a PTIME query class *may not* necessarily be Π-tractable. Indeed, Theorem 9 shows that $\mathsf{P} \neq \Pi\mathsf{T}^0_\mathsf{Q}$ unless $\mathsf{P} = \mathsf{NC}$.

(2) As shown by Corollary 6, any query class $\mathcal{Q}$ in P can be *made* Π-tractable by $\leqslant^{\mathsf{NC}}_{fa}$ reductions, which allow re-factorizations. In contrast, Theorem 9 tells us that when re-factorizations are not allowed, *i.e.,* when $\leqslant^{\mathsf{NC}}_F$ is used instead of $\leqslant^{\mathsf{NC}}_{fa}$, $\mathcal{Q}$ may not be transformed to a Π-tractable class of queries unless $\mathsf{P} = \mathsf{NC}$. In other words, a query class in P can be made Π-tractable only with *proper re-factorizations.*

We next prove Theorem 9.

Proof: To simplify the discussion we prove this *w.l.o.g.* for languages of pairs representing query classes. Consider the language S_{CVP} of pairs $\langle \varepsilon, q\rangle$, where q is an instance of the Circuit Value Problem (CVP), and ε is the empty string. That is, S_{CVP} is a language of pairs defined by CVP and a factorization $\Upsilon = (\pi_1, \pi_2, \rho)$ such that for any instance x of CVP, $\pi_1(x) = \varepsilon$, $\pi_2(x) = x$ and $\rho(\varepsilon, x) = x$. We show that unless $\mathsf{P} = \mathsf{NC}$, S_{CVP} is not Π-tractable and it may not be F-reduced to any Π-tractable language of pairs.

First assume by contradiction that S_{CVP} is Π-tractable. Then there exists a PTIME function $\Pi(\cdot)$ such that

$$\langle \Pi(\varepsilon), q\rangle \in \mathsf{NC}.$$

However, $\Pi(\varepsilon)$ is a constant function, and CVP is known to be P-complete under NC reductions [21]. From these and $\mathsf{NC} \subseteq \mathsf{P}$ [26] it follows that $\mathsf{NC} = \mathsf{P}$, which is widely open.

Now assume that there exists a Π-tractable language S of pairs such that $S_{\mathsf{CVP}} \leqslant_F^{\mathsf{NC}} S$. Then by Definition 7, there exist NC functions $\alpha(\cdot)$ and $\beta(\cdot)$ such that

$$\langle \varepsilon, q\rangle \in S_{\mathsf{CVP}} \text{ iff } \langle \alpha(\varepsilon), \beta(q)\rangle \in S.$$

Since S is Π-tractable, by Definition 1, there exists a PTIME function $\Pi_S(\cdot)$ such that

$$\langle D, Q\rangle \in S \text{ iff } \langle \Pi_S(D), Q\rangle \in \mathsf{NC}.$$

Putting these together, we have the following:

$$\langle \varepsilon, q\rangle \in S_{\mathsf{CVP}} \text{ iff } \langle \Pi_S(\alpha(\varepsilon)), \beta(q)\rangle \in \mathsf{NC}.$$

Again, $\Pi_S(\alpha(\varepsilon))$ is a constant function. Therefore, if $S_{\mathsf{CVP}} \leqslant_F^{\mathsf{NC}} S$, then we would have that $\mathsf{P} = \mathsf{NC}$. □

7.3 Complete Problems Under F-Reductions

Finally, we study complete problems for the set $\Pi\mathsf{T}_{\mathsf{Q}}^0$ of all Π-tractable query classes under F-reducibility. Along the same lines as Definition 6, we define the following notion.

Definition 8: A class $\mathcal{Q}$ of queries is *hard for* $\Pi\mathsf{T}_{\mathsf{Q}}^0$ *under F-reducibility* if $\mathcal{Q}' \leqslant_F^{\mathsf{NC}} \mathcal{Q}$ for all $\mathcal{Q}' \in \Pi\mathsf{T}_{\mathsf{Q}}^0$. It is *complete for* $\Pi\mathsf{T}_{\mathsf{Q}}^0$ *under F-reducibility* if it is Π-tractable and moreover, is hard for $\Pi\mathsf{T}_{\mathsf{Q}}^0$ under F-reducibility. □

It is rather hard to find a class of queries that is complete for $\Pi\mathsf{T}_{\mathsf{Q}}^0$ under F-reducibility. Indeed, the result below shows that the existence of such a complete query class is closely related to the open question whether $\mathsf{P} = \mathsf{NC}$.

Proposition 10: *Unless* $\mathsf{P} = \mathsf{NC}$*, a complete query class for* $\Pi\mathsf{T}_{\mathsf{Q}}^0$ *under F-reducibility is a witness in* $\mathsf{P} \setminus \mathsf{NC}$. □

Proof: Suppose that there exists a query class $\mathcal{Q}$ that is complete for $\Pi\mathsf{T}_{\mathsf{Q}}^0$ under F-reducibility. Then for all classes $\mathcal{Q}'$ of NC queries, $\mathcal{Q}' \leqslant_F^{\mathsf{NC}} \mathcal{Q}$, since by Definition 1, $\mathcal{Q}'$ is Π-tractable. Now consider a language S'_{CVP} of pairs $\langle q, \varepsilon\rangle$, where q is an instance of the Circuit Value Problem (CVP), and ε is the empty string (see the proof of Theorem 9). Obviously, S'_{CVP} is Π-tractable by Definition 1. Then by Definition 8, S'_{CVP} is F-reducible to the language of pairs for $\mathcal{Q}$. Moreover, $\Pi\mathsf{T}_{\mathsf{Q}}^0 \subseteq \mathsf{P}$, and CVP is P-complete. Putting these together, we have that unless $\mathsf{P} = \mathsf{NC}$, $\mathcal{Q}$ is in $\mathsf{P} \setminus \mathsf{NC}$. □

From Proposition 8 it follows that if we want to show $\mathsf{P} \neq \mathsf{NC}$ (resp. $\mathsf{P} = \mathsf{NC}$), we need to (a) find a query class $\mathcal{Q}$ that is complete for $\Pi\mathsf{T}_{\mathsf{Q}}^0$ under $\leqslant_F^{\mathsf{NC}}$, and moreover, (b) show that $\mathcal{Q}$ in not in NC (resp. is in NC).

In summary, the main result of this section is as follows: $\mathsf{NC} \subseteq \Pi\mathsf{T}_{\mathsf{Q}}^0 \subseteq \mathsf{P}$, and unless $\mathsf{P} = \mathsf{NC}$, $\Pi\mathsf{T}_{\mathsf{Q}}^0 \subset \mathsf{P}$, *i.e.,* P properly contains $\Pi\mathsf{T}_{\mathsf{Q}}^0$. Moreover, a complete query class for $\Pi\mathsf{T}_{\mathsf{Q}}^0$ under $\leqslant_F^{\mathsf{NC}}$ is a witness in $\mathsf{P} \setminus \mathsf{NC}$ unless $\mathsf{P} = \mathsf{NC}$.

8. Conclusion

This work is a first attempt to give a formal treatment of tractable queries on big data utilizing the preprocessing paradigm. We have proposed a notion of Π-tractable queries to characterize queries that are feasible on big data, in parallel polylog-time after a one-time PTIME preprocessing step off-line. Notions of decision problems and query classes that can be made Π-tractable are also introduced, based on factorizations of its instances into data sets and query classes. We have shown that several natural query classes are Π-tractable, and several query classes and decision problems can be made Π-tractable. We have also introduced two forms of NC reductions $\leqslant_{fa}^{\mathsf{NC}}$ and $\leqslant_F^{\mathsf{NC}}$, and shown that they are transitive, and are compatible with the set $\Pi\mathsf{T}_{\mathsf{Q}}$ of all query classes that can be made Π-tractable and with the set $\Pi\mathsf{T}_{\mathsf{Q}}^0$ of all Π-tractable query classes, respectively. In addition, we have shown that under $\leqslant_{fa}^{\mathsf{NC}}$, a natural problem is complete for the class $\Pi\mathsf{T}_{\mathsf{P}}$ of all decision problems that can be made Π-tractable. On the other hand, there exists a query class that is in P but is not in $\Pi\mathsf{T}_{\mathsf{Q}}^0$, and it is not reducible to any Π-tractable query class under $\leqslant_F^{\mathsf{NC}}$ unless $\mathsf{P} = \mathsf{NC}$.

The main conclusion of the paper is as follows:

(1) $\mathsf{NC} \subseteq \Pi\mathsf{T}_{\mathsf{Q}}^0 \subseteq \mathsf{P}$; and unless $\mathsf{P} = \mathsf{NC}$, $\Pi\mathsf{T}_{\mathsf{Q}}^0 \subset \mathsf{P}$, *i.e.,* not all PTIME queries are Π-tractable (Theorem 9).

(2) All query classes in P can be made Π-tractable by transforming them to a query class in $\Pi\mathsf{T}_{\mathsf{Q}}$ via $\leqslant_{fa}^{\mathsf{NC}}$ reductions (Theorem 5 and Corollary 6).

(3) For decision problems, $\mathsf{NC} \subseteq \Pi\mathsf{T}_{\mathsf{P}} = \mathsf{P}$ (Corollary 6).

That is, $\Pi\mathsf{T}_{\mathsf{Q}}^0$ is properly contained in P, unless $\mathsf{P} = \mathsf{NC}$. Nonetheless, all classes of PTIME queries can be made Π-tractable by means of proper factorizations.

As remarked earlier, $\Pi\mathsf{T}_{\mathsf{P}}$ allows arbitrary factorizations of problem instances into data and query parts, and corresponds to the set $\Pi\mathsf{T}_{\mathsf{Q}}$ of all query classes that *can be made* Π-tractable. In contrast, each query class in $\Pi\mathsf{T}_{\mathsf{Q}}^0$ *is* already Π-tractable with its predefined factorization (see Section 3).

In practice, the results help us determine whether a query class is feasible on big data. These also suggest an approach to answering a class $\mathcal{Q}$ of queries on big data: reduce $\mathcal{Q}$ to a $\Pi\mathsf{T}_{\mathsf{Q}}$-complete class that we know how to evaluate, by finding proper $\leqslant_{fa}^{\mathsf{NC}}$ reductions, if $\mathcal{Q}$ can be made $\Pi\mathsf{T}_{\mathsf{P}}$-tractable.

This work has raised as many questions as it has answered. A number of issues are open and require further investigation. We list some of the open issues below.

(1) As remarked in Section 1, we opt to use NC to define $\Pi\mathsf{T}_{\mathsf{Q}}^0$ because NC is the most prevalent model for parallel-time complexity, among other things. However, PRAM underlying NC may not be accurate for parallel systems such as MapReduce and its variants. These call for a full treatment of parallel computation models and parallel complexity classes that are more accurate than PRAM and NC, and moreover, take into account both computational cost and communication (coordination) complexity. Upon the availability of such a model and a complexity class, the class $\Pi\mathsf{T}_{\mathsf{Q}}^0$ of Π-tractable queries should then be revised accordingly.

(2) Another question concerns the existence of a complete query class for $\Pi\mathsf{T}_{\mathsf{Q}}^0$ under F-reducibility $\leqslant_F^{\mathsf{NC}}$. When NC-

factor reducibility $\leqslant_{fa}^{\mathsf{NC}}$ is considered, Theorem 5 tells us that the query class of one of BDS' factorizations is complete for $\Pi\mathsf{T}_{\mathsf{Q}}$. This result is useful since after all, we want to know what query classes can be *made* Π-tractable.

However, under $\leqslant_{F}^{\mathsf{NC}}$, the existence of a complete query class for the set $\Pi\mathsf{T}_{\mathsf{Q}}^{0}$ of all Π-tractable queries is closely related to whether $\mathsf{P} = \mathsf{NC}$ (Proposition 10), and remains *open*. This said, it does not hinder the study of $\Pi\mathsf{T}_{\mathsf{Q}}^{0}$, just like we still study (approximation) PTIME algorithms although we do not know whether $\mathsf{P} = \mathsf{NP}$.

(3) We have so far only considered Boolean queries and decision problems when studying Π-tractability. Nevertheless, Π-tractability for general queries, as well as for search problems and function problems, deserves a full treatment. In other words, Π-tractable functions and complexity classes consisting of such functions remain to be studied.

(4) We have studied two forms of reductions. At one end of the spectrum, $\leqslant_{F}^{\mathsf{NC}}$ is rather conservative in that no re-factorizations are allowed. At the other end, $\leqslant_{fa}^{\mathsf{NC}}$ is quite liberal in that the query and data parts can be combined and re-factorized. It is interesting to study other forms of reductions "between" $\leqslant_{F}^{\mathsf{NC}}$ and $\leqslant_{fa}^{\mathsf{NC}}$, which allow certain re-factorizations under practical constraints.

(5) Finally, there are a number of open issues in connection with query evaluation with preprocesssing. Given a problem that can be made Π-tractable, how can we *identify* a factorization that appropriately picks the dataset to be preprocessed, such that the problem instances can be solved in parallel polylog-time? If a given problem cannot be made Π-tractable, can we still preprocess its data set so that *approximate* parallel polylog-time algorithms can be developed for solving the problem? In addition, we have only presented preprocessing strategies that are commonly used in practice (Section 4). Other effective preprocessing methods are to be explored. For instance, under certain conditions, top-k query answering with early termination [14] may be made Π-tractable, which finds top-k answers in $Q(D)$ without computing the entire $Q(D)$. While it may not be easy to develop generic schemes to answer these questions, we envisage that in various application domains, effective techniques can be developed to make our queries Π-tractable on big data.

Acknowledgments. Fan is supported in part by the RSE-NSFC Joint Project Scheme, EPSRC EP/J015377/1, UK, and the 973 Program 2012CB316200 and NSFC 61133002 of China.

9. References

[1] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[2] F. N. Afrati, V. R. Borkar, M. J. Carey, N. Polyzotis, and J. D. Ullman. Map-reduce extensions and recursive queries. In *EDBT*, 2011.

[3] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, 2010.

[4] F. N. Afrati and J. D. Ullman. Transitive closure and recursive datalog implemented on clusters. In *EDBT*, 2012.

[5] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *J. Algorithms*, 57(2):75–94, 2005.

[6] P. Boldi, M. Rosa, M. Santini, and S. Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*, 2011.

[7] V. R. Borkar, M. J. Carey, and C. Li. Inside "big data management": ogres, onions, or parfaits? In *EDBT*, 2012.

[8] M. Cadoli, F. M. Donini, P. Liberatore, and M. Schaerf. Preprocessing of intractable problems. *Inf. Comput.*, 176(2):89–120, 2002.

[9] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan. On compressing social networks. In *KDD*, 2009.

[10] D. E. Culler, R. M. Karp, D. A. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, and T. von Eicken. Logp: A practical model of parallel computation. *Commun. ACM*, 39(11):78–85, 1996.

[11] A. Czumaj and C. Sohler. Sublinear-time algorithms. In *Property Testing*, pages 41–64, 2010.

[12] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1), 2008.

[13] R. Dorrigiv, A. López-Ortiz, and A. Salinger. Optimal speedup on a low-degree multi-core parallel architecture (LoPRAM). In *SPAA*, pages 185–187, 2008.

[14] R. Fagin, A. Lotem, and M. Naor. Optimal aggregation algorithms for middleware. *JCSS*, 66(4):614–656, 2003.

[15] W. Fan, J. Li, Z. Tan, X. Wang, and Y. Wu. Incremental graph pattern matching. In *SIGMOD*, 2011.

[16] W. Fan, J. Li, X. Wang, and Y. Wu. Query preserving graph compression. In *SIGMOD*, pages 157–168, 2012.

[17] T. Feder and R. Motwani. Clique partitions, graph compression and speeding-up algorithms. *JCSS*, 51(2):261–272, 1995.

[18] J. Fischer and V. Heun. Space-efficient preprocessing schemes for range minimum queries on static arrays. *SICOMP*, 40(2):465–492, 2011.

[19] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

[20] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[21] R. Greenlaw, H. J. Hoover, and W. L. Ruzzo. *Limits to Parallel Computation: P-Completeness Theory*. Oxford University Press, 1995.

[22] M. Grohe. The quest for a logic capturing ptime. In *LICS*, pages 267–271, 2008.

[23] A. Y. Halevy. Answering queries using views: A survey. *VLDB J.*, 10(4):270–294, 2001.

[24] D. Harnik and M. Naor. On the compressibility of NP instances and cryptographic applications. *SICOMP*, 39(5):1667–1713, 2010.

[25] J. M. Hellerstein. The declarative imperative: experiences and conjectures in distributed logic. *SIGMOD Record*, 39(1):5–19, 2010.

[26] D. S. Johnson. A catalog of complexity classes. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 67–161. The MIT Press, 1990.

[27] N. D. Jones. An introduction to partial evaluation. *ACM Comput. Surv.*, 28(3):480–503, 1996.

[28] H. J. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *SODA*, pages 938–948, 2010.

[29] P. Koutris and D. Suciu. Parallel evaluation of conjunctive queries. In *PODS*, pages 223–234, 2011.

[30] M. Lenzerini. Data integration: A theoretical perspective. In *PODS*, 2002.

[31] H. Maserrat and J. Pei. Neighbor query friendly compression of social networks. In *KDD*, 2010.

[32] S. Navlakha, R. Rastogi, and N. Shrivastava. Graph summarization with bounded error. In *SIGMOD*, 2008.

[33] C. H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

[34] R. Ramakrishnan and J. Gehrke. *Database Management Systems*. McGraw-Hill Higher Education, 2000.

[35] G. Ramalingam and T. Reps. On the computational complexity of dynamic graph problems. *TCS*, 158(1-2), 1996.

[36] D. Ron. Algorithmic and analysis techniques in property testing. *Foundations and Trends in Theoretical Computer Science*, 5(2):73–205, 2010.

[37] D. Saha. An incremental bisimulation algorithm. In *FSTTCS*, 2007.

[38] G. Santos. SSD ranking: The fastest solid state drives. *http://www.fastestssd.com/featured/ssd-rankings-the-fastest-solid-state-drives/#pcie*, Oct 2012.

[39] D. Suciu and V. Tannen. A query language for NC. *J. Comput. Syst. Sci.*, 55(2):299–321, 1997.

[40] L. G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.