Queries are easier than you thought*(probably)

Serge Abiteboul† Kevin Compton‡ Victor Vianu§

Abstract

The optimization of a large class of queries is explored, using a powerful normal form recently proven. The queries include the *fixpoint* and *while* queries, and an extension of *while* with arithmetic. The optimization method is evaluated using a probabilistic analysis. In particular, the average complexity of *fixpoint* and *while* is considered and some surprising results are obtained. They suggest that the worst-case complexity is sometimes overly pessimistic for such queries, whose average complexity is often much more reasonable than the provably rare worst case. Some computational properties of queries are also investigated. A probabilistic notion of *boundedness* is defined, and it is shown that all programs in the class considered are bounded almost everywhere. An effective way of using this fact is provided.

1 Introduction

The complexity of query languages has traditionally been investigated using worst-case bounds. We argue that this approach provides an overly pessimistic picture of query complexity. We provide here a *probabilistic* analysis based on techniques from 0-1 laws [Fag90]. We focus on the optimization of queries resulting from the normal form of [AV91b]. The normal form reduces the computation on an input instance to another computation on an ordered instance. We prove that the expected size of the ordered instance is much smaller than that of the original input. Using this, we investigate the average complexity of queries, and obtain several surprising results. These results highlight the potential for optimization provided by the normal form.

The focus is on a large class of queries which includes the most commonly studied extensions of relational calculus (FO) such as Datalog$^\neg$, and the *fixpoint* and *while* queries. All the languages share the property that they can be viewed as a computing device interacting with a database using a fixed set of FO queries. The most powerful of these languages provides full computational power "loosely coupled" with the database by the FO queries. This can be formalized, for instance, as *while* augmented with arithmetic, introduced in [CH80] and denoted by *while*$^+$. This language is, in some sense, the formal counterpart of database application languages such as C+SQL which embed a relational language into a full programming language.

The basis of the present work is a normal form for *while*$^+$ shown in [AV91b]. This normal form demonstrates that the computation of a *while*$^+$ program is entirely determined by a certain equivalence relation between tuples, which is computable by a *fixpoint* query. Intuitively, the equivalence relation reflects the limited ability of the program to discern among pieces of the input using the finite set of calculus queries at its disposal. Once the equivalence relation is obtained, the input can be simplified by replacing each equivalence relation by a single integer. As shown here, the cardinality of the equivalence relation may be much smaller than that of the input. This opens up a new avenue for optimization. To evaluate its potential impact, we use a probabilistic analysis. The analysis shows that the expected gain in the optimization is indeed high, because the expected number of equivalence classes is low. For the analysis, we make the important assumption that all inputs are equally likely. Although this is unrealistic in many

*Proceedings of ACM SIGMOD-SIGACT-SIGART on Principles of Database Systems, San Diego, June 1992

†I.N.R.I.A., 78153 Le Chesnay, France. abitebou@inria.inria.fr

‡EECS Dept., University of Michigan, Ann Arbor, MI 48109-2122, kjc@eecs.umich.edu, visiting I.N.R.I.A.

§Univ. of California at San Diego, CSE C-0114, La Jolla, CA 92093-0114, vianu@cs.ucsd.edu, visiting I.N.R.I.A. Work supported in part by the NSF under grants IRI-8816078 and INT-8817874.

applications, it allows us to demonstrate the potential of computing with the normal form. Also, we show that the results carry through under modified assumptions, including sparsity of the input. The study of more refined hypothesis on the input is left for future research.

Using the same probabilistic approach, we look at an important computational property related to optimization, namely *boundedness* (see [KA89]). In a language with recursion, boundedness of a program says that the program terminates after a constant number of iterations, independent of the data. In other words, the program is equivalent to a FO query. Boundedness is undecidable in general. We define a notion of *probabilistic boundedness* as follows. Informally, a program p is "bounded almost everywhere" iff there exists a FO formula ϕ such that p is equivalent to ϕ with very high probability. We prove the surprising fact that *every* terminating *while*$^+$ program is bounded almost everywhere. More importantly ¿from a practical standpoint, for a specific input I, one can test if ϕ yields the correct answer using another first-order query ψ. Both ϕ and ψ can be effectively constructed from the program p.

The results on boundedness are closely related to work by Kolaitis and Vardi on 0-1 laws for infinitary first-order logic with k variables, denoted $L^k_{\infty\omega}$ [KV87, KV90a, KV92]. The boundedness of *while*$^+$ almost everywhere follows from their results. However, the construction of ϕ above from p is non-effective in the general case considered there. An effective construction can be obtained by specializing to *while*$^+$ a quantifier elimination technique developed by Grandjean [Gra83] and extended to $L^k_{\infty\omega}$ in [KV92] (see Remark 5.1 below).

The results on boundedness show that terminating *while*$^+$ queries are easy on most inputs, although they can become arbitrarily hard on some. This naturally leads to considering the *average complexity* of queries. We look at the average sequential and parallel complexity of the *fixpoint* queries and show that:

- the average sequential complexity of *fixpoint* is LOGSPACE;
- the average parallel complexity of *fixpoint* is constant time with polynomially many processors (AC_0).

Note the difference with the worst-case complexity of *fixpoint*, which is PTIME.

For *while*, the situation is slightly more complicated. We are able to prove an average sequential space complexity of LOGSPACE. We show that there are PSPACE-complete *while* query whose average complexity is PTIME. This cannot be generalized to arbitrary *while* queries because for some of these queries, the cost for the worst case instances is sufficiently high to outweigh the PTIME cost required by most instances.

The fact that the uniform complexity measure may be accompanied by low average-case complexity is well known in certain cases. For example, it is shown in [Wil84] that the average-case time complexity of k-colorability is constant. While most such work assumes a probability measure giving equal weights to all inputs, the average-case behavior of query optimization techniques under variable probability measures on directed graphs is considered in [SN91].

A main goal of the paper is to demonstrate the potential of computating with the normal form. Indeed, efforts are currently devoted to turn the normal form technique into a practical algorithm [AG92]. The key idea is to compute with the standard algorithm and (with little overhead) keep track of informations concerning the normal form. This allows to save work by slowly moving to the normal form computation when possible without the large overhead of computing it ahead of time. Some measures obtained with a Prolog prototype already confirm the potential of the technique.

2 A Large Class of Queries

In this section, we review briefly the main query languages of interest in the paper. We start with the simplest ones, and build up to *while*$^+$.

To begin, we review informally some terminology and notation of relational databases and query languages. A general presentation of the database field is given in [Ull88], and of database theory in [Kan91].

A *database schema* **S** is a finite set of predicates of fixed arity (denoted $P, Q, R, ...$). A *relational database* over **S** is a finite structure over the predicates in **S**. A *query* ϕ is a mapping from finite structures over an *input* schema to finite structures over an *output* schema. The mapping must be computable and *generic*: for each automorphism ν of an input I, $\phi(\nu(I)) = \nu(\phi(I))$. A query language or computing device is called *complete* if it expresses all queries. Complexity classes of queries are defined based on Turing complexity in the natural fashion. For example, the PTIME queries consist of all generic mappings ϕ such that there is a TM which, given on the tape a standard encoding $enc(I)$ of an input I, produces a standard encoding of $\phi(I)$, in time polynomial in $|enc(I)|$.

We assume familiarity with the first-order queries, FO, expressed by the relational calculus and algebra. Recall also that FO is in (uniform) AC_0 [Imm87], so FO queries can be evaluated in constant parallel time with polynomial resources.

There are many useful queries that FO cannot express, such as the transitive closure of a graph. Numerous extensions of FO with recursion have been proposed. Most of them converge towards two central classes of queries: *fixpoint* and *while* [Cha81, CH82]. Both allow inductive definitions of relations up to a fixpoint, by iterating a FO formula. For the *fixpoint* queries, results of iterations are accumulated, so convergence is guaranteed in polynomial time. For *while*, each iteration is destructive, so *while* $\subseteq$ PSPACE. In particular, *while* queries may not converge. We describe *fixpoint* and *while* next.

The Fixpoint Queries The *fixpoint queries* (*fixpoint*) [CH82] are constructed using the first-order constructors as in *FO* together with a fixpoint operator (μ). The fixpoint operator binds a predicate symbol R that is free and that appears only positively (i.e., under an even number of negations) in the formula. The semantics is given by the least fixpoint of the formula.

The same expressive power can be achieved using an *inflationary fixpoint logic* (*FO+IFP*) [GS86]. The inflationary fixpoint operator (*IFP*) no longer requires that R occur positively in $\phi(R)$, and convergence is instead guaranteed in polynomial time by accumulating iterations of $\phi(R)$ up to a fixpoint. Inflationary fixpoint formulas are defined next.

Inflationary fixpoint formulas are obtained by repeated applications of first-order operators ($\neg, \vee, \wedge, \exists, \forall$) and the inductive fixpoint operator starting from atoms. The inductive fixpoint operator is defined as follows. Let ϕ(T) be an *FO+IFP* formula with n free variables, where T is an n-ary predicate occurring in ϕ. Then $IFP(\phi(T), T)$ denotes the n-ary predicate which is the limit of the sequence defined by: $J_0 = T$ and for each $i > 0$, $J_i = \phi(J_{i-1}) \cup J_{i-1}$. If $\vec{t}$ is a sequence of n variables or constants, $IFP(\phi(T), T)(\vec{t})$ is a formula.

Other languages expressing *fixpoint* are various extensions of Datalog with negation (Datalog$^\neg$). For example, Datalog$^\neg$ with inflationary semantics [AV91a, KP88], and with well-founded semantics [Gel89], both express *fixpoint*. Datalog$^\neg$ with stratified semantics is strictly within *fixpoint* [Dah87, Kol91].

The While Queries The *while* queries also allow an inductive definition of relations up to a fixpoint. They were originally introduced in a procedural form, under the names *LE* in [Cha81], and *RQ* in [CH82]. Relational algebra is extended with (i) sorted relational variables $(X, Y, ...)$, (ii) assignment of FO queries to variables, and (iii) a *while* construct allowing to iterate a program while some first-order condition (e.g., $X = \emptyset$) holds. It is clear that *while* programs may not terminate.

The *while* language can also be seen as first-order logic extended with a partial fixpoint operator PFP [AV91a], yielding *partial fixpoint logic, FO+PFP*. The difference with $FO + IFP$ lies in the definition of the PFP operator, which is no longer inflationary. The interpretation of $PFP(\phi(T), T)$ is the following: $PFP(\phi(T), T)$ denotes the predicate which is the limit, **if it exists**, of the sequence defined by: $J_0 = T$ and for each $i > 0$, $J_i = \phi(J_{i-1})$ (if ϕ is undefined on J_{i-1}, then J_i and the interpretation of $PFP(\phi(T), T)$ are undefined). The partial fixpoint of $\phi(R)$ is computed by iterating ϕ destructively.

It is shown in [AV91a] that *partial fixpoint logic* expresses precisely the *while* queries.

As we have seen, there are many different languages which express the *fixpoint* and *while* queries. In the paper, we will refer to a *fixpoint* (or *while*) program, meaning a program in any of the languages expressing *fixpoint* (or *while*).

Expressiveness of Fixpoint and While The expressive power of *fixpoint* and *while* is well understood with the assumption that an order is available. A database is said to be *ordered* if one particular binary relation *succ* provides a successor relation on the constants occurring in the database. In this case, *fixpoint* expresses PTIME [Imm86, Var82], and *while* expresses PSPACE [Var82]. However, in the absence of order neither can express the query *even* on a set[1] [CH82].

Due to the expressiveness with order, *fixpoint* and *while* are distinct if PTIME $\neq$ PSPACE. It was shown recently in [AV91b] that indeed the order assumption can be removed and *fixpoint* = *while* iff PTIME = PSPACE. Further connections between fixpoint logics and classical complexity classes are obtained in [AVV91].

While$^+$ The *while* queries are characterized by the fact that they manipulate the input using some finite set of FO queries. They can be viewed as the result of coupling a finite set of FO queries with a finite-state control. More generally, one can extend first-order logic with arbitrary computing capability. This idea was suggested in [CH80] and pursued in [AV91b]. In [CH80], this was achieved by extending *while* with (i) integer variables, (ii) an increment and decrement operation, and (iii) loops whose termination condition is of the form $i = 0$, where i is an integer variable. We denote this language (and, by extension, the class of queries it expresses) by *while*$^+$. The class of *terminating while*$^+$ programs is denoted by *while*$^+\downarrow$. In [AV91b], this type of computation is captured by a computational device called *Loosely Coupled Generic Machine*, GM^{loose}. GM^{loose} extends the finite-state control of *while* to a computationally complete device. It consists of a Turing Machine (TM) augmented with a finite set of fixed-arity relations forming a *relational store*. Both *while*$^+$ and GM^{loose} model accurately database application programs, which often use a FO relational language

[1] The query *even* is defined by *even(R)* = **true** iff $|R|$ is even.

(say, SQL) embedded in a full programming language (say, C). The TM part of the machine models the use of C, whereas the *FO* queries on the relational store correspond to the SQL queries.

GM^{loose} and *while*$^+$ express the same class of queries. They are of special interest because they subsume most query languages which have emerged as central, including the *fixpoint* and *while* queries. Although *while*$^+$ provides arbitrarily powerful computation outside the database, the manipulation of the databases is done using just some finite set of *FO* queries. This is a source of limitation in expressive power, which varies widely on ordered vs. unordered inputs. Indeed, *while*$^+$ is complete on ordered inputs but collapses to *FO* on unordered sets. In particular, it cannot express the *even* query.

3 A Normal Form

In [AV91b], a normal form for *while*$^+$ queries is shown, which yields new insight into the computation performed by such queries and is a crucial tool in this paper. The normal form provides a bridge between computation without order and computation with order. It says, intuitively, that each *while*$^+$ computation over an unordered domain can be reduced to a *while*$^+$ computation over an *ordered* domain via a *fixpoint* query. In the context of *while*$^+$, this means that the computation is transferred entirely to the integer domain after the *fixpoint* query.

More precisely, a *while*$^+$ program in the normal form consists of two phases. The first is a *fixpoint* query which performs an analysis of the input. It computes an equivalence relation on tuples which is a "congruence" with respect to the rest of the computation, in that equivalent tuples are treated identically throughout the computation. Thus, each equivalence class is treated as an indivisible "block" of tuples, which is never split later in the computation. The *fixpoint* query outputs these equivalence classes in some order, so that each class can then be thought of abstractly as an integer.

The second phase consists of a *while*$^+$ query which can be viewed as computing on an *ordered* database obtained by replacing each equivalence class produced in the analysis phase by its corresponding integer.

The normal form reflects the fact that *while*$^+$ queries have limited ability to distinguish among tuples in their input. This is because they do not have full access to their input, but only interact with it using some fixed set of *FO* queries. Tuples which cannot be distinguished by those queries can never be distinguished throughout the computation. More precisely, a *while*$^+$ program p is called k-ary if the maximum number of variables in an *FO* query in p (expressed in the calculus) is k. For a k-ary program p and k-ary tuples u and v, we say that p cannot distinguish between u and v on input I if for every relation R in p, $\pi_R(u)$ is in R precisely when $\pi_R(v)$ is in R. Let $\equiv_p$ denote this equivalence relation among k-tuples. The normal form is based on the fact that $\equiv_p$ can be computed by a *fixpoint* query for each p. Furthermore, the *fixpoint* query converges in a number of stages which is polynomial in the size of $\equiv_p$. For an input I and a program p, let the *p-size of I*, denoted $|I|_p$, be the number of equivalence classes of $\equiv_p$. The normal form allows a more accurate view of the complexity of *while*$^+$ queries. Indeed, it says that each such query can be written as $f \circ w$ where

- f is a *fixpoint* query which iterates a *FO* formula $P(|I|_p)$ times, where P is a polynomial; and
- w is a *while*$^+$ program on an *ordered* input of size $|I|_p$.

For instance, each *while* query can be evaluated in two phases. The first is a *fixpoint* computation whose time complexity is polynomial in $|I|$. This is followed by a computation whose space complexity is polynomial in $|I|_p$. When $|I|_p$ is much smaller than $|I|$, the use of the normal form yields an optimized computation. The use of the normal form for optimization has not been previously explored and is central to the present paper.

4 Expected P-Size of Inputs

In the normal form, a computation with an input of size $|I|$ is replaced by a (no more complex) computation on an input of size $|I|_p$, at the cost of a *fixpoint* query. If $|I|_p$ is much smaller than $|I|$, the new computation is on a much smaller input, which lowers both the space and time complexity. A natural question is then: "what are the chances that $|I|_p$ is much smaller than $|I|$?".

To see that this is often the case, it is useful to note the connection between automorphisms of the input and $\equiv_p$. If there is an automorphism of the input mapping u to v, then u and v cannot be distinguished by p, so $u \equiv_p v$. Thus, let $\equiv$ be defined by $u \equiv v$ iff $h(u) = v$ for some automorphism h of the input. It follows from the above that $\equiv$ refines $\equiv_p$, and $|I|_p \leq |\equiv|$.

For instance, consider the following special cases:

- if I is a set, then $|\equiv|$ is a constant, so $|I|_p$ is constant; and
- if I is a complete binary tree, then $|\equiv| \leq P(log(|I|))$ for some polynomial P, so $|I|_p \leq P(log(|I|))$.

In both cases, $|I|_p$ is much smaller than $|I|$ and the potential for space and time optimization is high. Clearly, this is not the case for all inputs. For instance, if I is a graph consisting of one simple path, then $|I|_p$ could equal $P(|I|)$ for some polynomial P. However, such cases are rare, assuming that all inputs are equally likely. Indeed, this is confirmed by an analysis of the expected size of $|I|_p$ which we provide next. The analysis is used later in the paper to show that the average complexity for many queries is much lower than suggested by the known worst case complexity.

We first demonstrate that for a given k, and programs p of arity k, the p-size is bounded for most instances. Recall that a program of arity k uses only FO formulas with at most k variables. For each k and each input I, let $\equiv_k$ be the equivalence relation over k-tuples with entries in I defined by: for each k-tuples u, v, $u \equiv_k v$ iff for each FO formula ϕ using k variables, $u \in \phi(I) \Leftrightarrow v \in \phi(I)$. The *$k$-size* of I, denoted $|I|_k$, is the number of equivalence classes in $\equiv_k$. Observe that for each program p of arity k and each I, $\equiv_k$ refines $\equiv_p$ so $|I|_p \leq |I|_k$.

Consider any property P of inputs. Let $\mu_n(P)$ be the fraction of inputs with n constants which satisfy P. We say that P *holds almost everywhere* iff

$$\lim_{n\to\infty} \mu_n(P) = 1.$$

For instance, we say that two formulas ϕ and ψ are equivalent almost everywhere, if

$$I \models \phi \text{ iff } I \models \psi$$

holds almost everywhere.

The result of this section shows that the p-size of most inputs is a *fixed* constant, independent of the size of the input. In some sense, this is the best that one could expect.

Theorem 4.1 For each schema $\mathbf{R}$ and each *while*$^+$ program p, there exists some constant M such that $|I|_p \leq M$ almost everywhere.

Proof: (sketch) Suppose p has arity k. We will show that for some M, $|I|_k \leq M$ almost everywhere, i.e.

$$\lim_{n\to\infty} \mu_n(|I|_k \leq M) = 1.$$

This then implies that $|I|_p \leq M$ almost everywhere.

To show this, we use the so called "extension axioms", introduced in conjunction with 0-1 laws. For simplicity, we describe extension axioms for inputs which are directed graphs (binary relations). This can be extended to inputs of arbitrary arity. Let k be fixed. The k-ary extension axioms, denoted $\mathcal{A}_k$, are sentences stating that any subgraph of k nodes can be extended by an additional node in all possible ways. More precisely, for each k, $\mathcal{A}_k$ contains all sentences of the form
$\forall x_1 ... \forall x_k ((\bigwedge_{i\neq j}(x_i \neq x_j))$
$\Rightarrow \exists y (\bigwedge_i (x_i \neq y) \wedge connections(x_1, ..., x_k; y)))$

where $connections(x_1, ..., x_k; y)$ is a conjunction of literals containing, for each x_i, one of $G(x_i, y)$ or $\neg G(x_i, y)$, and one of $G(y, x_i)$ or $\neg G(y, x_i)$. For example, for $k = 3$, one of the 4^3 sentences of the form $connections(x_1, x_2, x_3; y)$ is

$$G(x_1, y) \wedge \neg G(y, x_1) \wedge \neg G(x_2, y) \wedge$$
$$\neg G(y, x_2) \wedge G(x_3, y) \wedge G(y, x_3).$$

Let I be an input graph satisfying all sentences in $\mathcal{A}_k$. Then it can be shown that

(†) for each k-ary tuple u, the equivalence class of u in $\equiv_k$ is determined *locally*, by the equalities and edges among the coordinates of u.

The proof of (†) is non-trivial. It involves a k-pebble game which characterizes the ability to distinguish among k-ary tuples using FO formulas with k variables [KV90a].

Since there are finitely many distinct patterns of equalities and edges among k coordinates, (†) implies that there is a fixed M such that $|I|_k \leq M$ for each $I \models \mathcal{A}_k$. Lastly, we show that "most" inputs satisfy $\mathcal{A}_k$. Indeed, it is easily verified that for each $\sigma \in \mathcal{A}_k$,

$$\mu_n(I \not\models \sigma) \leq n^k (1 - \frac{1}{4^k})^{n-k},$$

so $\lim_{n\to\infty} \mu_n(I \not\models \sigma) = 0$ and $\lim_{n\to\infty} \mu_n(I \models \sigma) = 1$. Since there are finitely many sentences in $\mathcal{A}_k$, it follows that $\lim_{n\to\infty} \mu_n(I \models \mathcal{A}_k) = 1$. Thus, $\lim_{n\to\infty} \mu_n(|I|_k \leq M) = 1$ and $\lim_{n\to\infty} \mu_n(|I|_p \leq M) = 1$, so $|I|_p \leq M$ almost everywhere. □

Thus, Theorem 4.1 says that, in many cases, the p-size of the input is much smaller than its size. In such cases, the normal form results in substantial optimization.

5 Probabilistic Boundedness

The queries in *while*$^+$ display a wide range of expressive power. For example, if the input is a set (unary relation), then any *while*$^+\!\downarrow$ program is equivalent to a first-order formula, and so is bounded. It turns out that *while*$^+\!\downarrow$ programs are in fact bounded on "most" inputs, even those which are not sets. How can this be? Intuitively, this is because, by the normal form, *while*$^+\!\downarrow$ computations can be reduced to computations on inputs of size $|I|_p$; and we proved that for "most" inputs, $|I|_p$ is a constant.

To formalize the above, we define a probabilistic notion of boundedness.

Definition: A program p is *bounded almost everywhere* if there exists a FO query ϕ such that p is equivalent to ϕ almost everywhere.

We now show:

Theorem 5.1 Each *while*$^+\downarrow$ program is bounded almost everywhere.

Proof: (sketch) Let p be a terminating *while*$^+$ program. For each m, consider the family $\mathcal{I}_m$ of instances I over $\mathbf{R}$ for which $|I|_p \leq m$. By the normal form, p can be reduced to a query on an input of size $|I|_p$ via a *fixpoint* query which iterates a FO formula a number of times which is polynomial in $|I|_p$. It follows that for each fixed m, there exists a FO query ϕ_m such that $\phi_m(I) = p(I)$ for each $I \in \mathcal{I}_m$. On the other hand, we have shown that there is some fixed M such that $|I|_p \leq M$ almost everywhere. It follows that p is equivalent to ϕ_M almost everywhere, so p is bounded almost everywhere. $\Box$

The probabilistic boundedness shown above says that for any given *while*$^+\downarrow$ program p, there exists a FO query ϕ such that, on almost all inputs, one can just apply ϕ instead of running p. However, when faced with a *particular* input, one has to decide whether to use ϕ or p; knowing that one can apply ϕ almost everywhere is not a priori of much practical help. We next resolve this problem by showing that it is in fact easy to test whether ϕ can be applied instead of p. Indeed, the test involves just another FO query ψ. Moreover, both ϕ and ψ are effectively computable from p.

Theorem 5.2 For every *while*$^+\downarrow$ program p, there exist FO queries ϕ and ψ, effectively computable from p, such that:

1. if $I \models \psi$ then $p(I) = \phi(I)$; and,
2. ψ holds almost everywhere. $\Box$

Proof: (sketch) For ψ, one can simply take $\psi = \bigwedge \mathcal{A}_k$, and ψ is effectively computable from p (in fact, it only depends on k). The effective construction of ϕ uses the normal form. It is based on the fact that, for inputs I such that $|I|_p \leq M$, there are finitely many possible outcomes to the first phase of the normal form (the *fixpoint* query). $\Box$

The above theorem suggests the following procedure *speed.p* for evaluating a *while*$^+\downarrow$ program p on input I:

procedure *speed.p*;
if $I \models \psi$ then (i) apply ϕ
otherwise (ii) run p on I.

Theorem 5.2 guarantees that (i) will almost always be used, so on almost all inputs p is reduced to applying two FO queries.

The previous analysis fails for non-terminating *while*$^+$ programs, since we cannot detect termination even for instances satisfying the extension axioms. The problem of termination disappears in the case of restricted languages like *while*, where termination can be tested [AS91].

Remark 5.1 The results in this section are closely related to recent work of Kolaitis and Vardi on 0-1 laws [KV87, KV90a, KV92]. They state a *non-effective* version of Theorem 5.2 for formulas in *infinitary first-order logic with k variables*[2], denoted $L^k_{\infty\omega}$. More precisely, they show that for each $L^k_{\infty\omega}$ formula ξ there exist first-order queries ϕ and ψ such that ψ holds in almost all finite structures and wherever ψ holds, ξ and ϕ are equivalent. Their result implies that every *while*$^+$ program p is also bounded almost everywhere, since it can be shown that any such program can be phrased in terms of $L^k_{\infty\omega}$ queries.

The results in [KV87, KV90a, KV92] apply to $L^k_{\infty\omega}$ and are therefore non-constructive. The definition of ϕ above from ξ uses a quantifier elimination technique first introduced in [Gra83] and extended to $L^k_{\infty\omega}$ in [KV90b]. From the proof of that result [KV92], it can be seen that the quantifier elimination technique can be made effective when applied to *while*$^+$. Thus, as pointed out by Vardi [Var92], this can be used to obtain an alternative proof of Theorem 5.2. Note that *while*$^+$ can be viewed as the *effective fragment* of $L^k_{\infty\omega}$, as discussed in [AVV92].

6 Average Complexity of Fixpoint and While

We have seen that each *while*$^+\downarrow$ program is equivalent on almost all instances to a FO query. The cost of evaluating the program on such instances is low, and the number of the other (potentially expensive) instances is small. This suggests that for some programs the cost of the expensive instances may be amortized by the low cost on most instances. Then the average complexity of evaluating such programs could be much lower than the rare worst case complexity, and would provide a more accurate picture of the complexity of evaluating such programs. Clearly, this is not the case for *while*$^+\downarrow$ programs in general, which can be made arbitrarily hard on the expensive instances. However, we show such results for restricted but important languages such as *fixpoint* and *while*.

We start by making more precise our notion of average complexity. Let p be a *while*$^+\downarrow$ program. Suppose that c_p is a time or space cost function associated with the evaluation of p on a given input I. The *average* value of c_p on inputs of size n is defined

[2]The infinitary logic $L^k_{\infty\omega}$ allows infinite conjunctions and disjunctions of FO formulas.

as

$$\bar{c}_p(n) = \frac{1}{2^{n^2}} \sum_{|I|=n} c_p(I),$$

where here it is assumed inputs are binary relations. (Note that there are 2^{n^2} binary relations with n constants). Let f be a function of n. Then p *has average complexity* f with respect to the cost function c_p if $\bar{c}_p(n) = O(f(n))$. A query q has average complexity f if there exists a program p expressing q, whose average complexity is f. For a class $\mathcal{F}$ of functions, a query has average complexity $\mathcal{F}$ if it has average complexity f for some $f \in \mathcal{F}$. For example, a query has PTIME average complexity if it has $P(n)$ average complexity for some polynomial P. Note that the above definitions apply to any complexity measure, sequential or parallel. Average complexity based on combined time/space measures (such as AC_0, constant parallel time with polynomially many processors) is defined similarly.

The notion of average complexity presented here is the standard one in analysis of algorithms. It might be termed the *asymptotic approach* because it emphasizes complexity as input size increases. There is another approach due to Levin [Lev86] (see Gurevich [Gur91] for a full development) which, in some sense, averages over all finite inputs. We have not investigated query expectations using this alternative approach, which also appears to raise interesting questions.

We first provide the results on the average complexity of *fixpoint*, then turn to *while*.

Theorem 6.1 Each *fixpoint* query has average complexity (i) LOGSPACE (sequential space) and (ii) AC_0 (parallel time/space).

Proof: (sketch) Let p be a *fixpoint* program. By Theorem 5.2, there exist FO formulas ϕ and ψ such that

$$I \models \psi \Rightarrow p(I) = \phi(I),$$

and $I \models \psi$ almost everywhere. Consider the procedure *speed.p*. Let $c_{speed.p}$ denote its costs and c_p denote the cost of the straightforward algorithm. Then,

- if $I \models \psi$, $c_{speed.p}(I) = c_{\psi;\phi}(I)$,
- if $I \not\models \psi$, $c_{speed.p}(I) = c_{\psi;p}(I)$

where ";" is used for composition, so

$$c_{\psi;\phi}(I) = c_{\psi}(I) + c_{\phi}(I), \text{ and}$$
$$c_{\psi;p}(I) = c_{\psi}(I) + c_p(I).$$

Let k be the arity of p. From the proofs of Theorems 5.1 and 5.2, it follows that all instances satisfying the extension axioms $\mathcal{A}_k$ also satisfy ψ. By the analysis in the proof of Theorem 5.1,

$$(\dagger)\ \mu_n(I \not\models \psi) \leq 4^k n^k (1 - \frac{1}{4^k})^{n-k}$$

(the factor 4^k is due to the fact that there are 4^k extension axioms in $\mathcal{A}_k$). Then

$$\begin{aligned}(\ddagger)\quad \bar{c}_{speed.p}(n) \leq\ & \mu_n(I \models \psi) \cdot \max_{|I|=n}\{c_{\psi;\phi}(I)\} \\ & + \mu_n(I \not\models \psi) \cdot \max_{|I|=n}\{c_{\psi;p}(I)\}.\end{aligned}$$

Consider now (i), i.e., the sequential space complexity. Then

$$\max_{|I|=n} c_{\psi;\phi}(I) \leq K \cdot \log(n) \quad \text{and} \quad \max_{|I|=n} c_{\psi;p}(I) \leq P(n)$$

for some constant K and some polynomial P. From ($\dagger$) and ($\ddagger$), it then follows that

$$\begin{aligned}\bar{c}_{speed.p}(n) \leq\ & [1 - 4^k n^k (1 - \tfrac{1}{4^k})^{n-k}] \cdot K \cdot \log(n) + \\ & 4^k n^k (1 - \tfrac{1}{4^k})^{n-k} P(n).\end{aligned}$$

Since $\lim_{n \to \infty} 4^k n^k (1 - \frac{1}{4^k})^{n-k} P(n) = 0$, it follows immediately that $\bar{c}_p(n) = O(\log(n))$. Part (ii) is proven similarly. □

Note that Theorem 6.1 is likely to provide also an improvement over the worst case sequential time complexity of the fixpoint queries, since it is conjectured that LOGSPACE $\subset$ PTIME. However, the improvement is more striking in (ii), where parallel complexity is considered.

We next look at the average complexity for *while*. The approach is similar to the above. Since the worst-case space complexity of *while* is polynomial, the same analysis as above shows that the average sequential space complexity of *while* is LOGSPACE.

Theorem 6.2 Terminating *while* queries have LOGSPACE average complexity. □

The analysis for the average sequential time complexity and parallel time/space complexity does not carry through from *fixpoint*. This is because the time complexity for *while* programs is exponential for the "expensive" instances, i.e. those which do not satisfy $\mathcal{A}_k$.

One may legitimately wonder if there are "hard" *while* queries whose average time complexity is significantly better than the worst case time complexity. We next show that indeed there are such queries.

Proposition 6.3 There exists a PSPACE-complete *while* query whose average sequential time complexity is PTIME.

Proof: (sketch) One can devise a PSPACE-complete *while* query with PTIME average complexity, starting from any PSPACE-complete problem naturally represented as a structure. For instance, any of the problems that ask about winning strategies for games on graphs will do (see Garey and Johnson [GJ79]). For example, consider Generalized Hex, where an instance is a graph and two distinguished vertices. Players I and II alternate placing pebbles on nondistinguished vertices. Player I wins by forming a path with his pebbles between the two distinguished vertices; Player II wins if she prevents this. The problem is to determine if Player I has a winning strategy. It is easy to see that an exhaustive search of the game tree for this problem can be done in time $n!$.

Since this problem is PSPACE-complete it may be expressed by a *while* query on the graph, together with a successor relation *succ* on the constants. Note that one can apply the query on unordered inputs, by screening out the inputs for which *succ* is not a successor relation (this is a FO operation), then asking whether Player I has a winning strategy on the graph with the two distinguished elements. This can be expressed by a *while* query. There is still a difficulty because the $n!$ running time to analyze the game strategy in the worst case gives exponential expected time. This can be solved by padding. The problem is modified to determine if Player I has a winning strategy on a graph whose vertices are among the first $\sqrt{n}$ elements in *succ* (n is the number of constants in *succ*). We can talk about the first $\sqrt{n}$ elements since addition and multiplication can be defined on any ordered structure using just fixpoint logic. It is easy to see that the query is still PSPACE-complete, but now the expected time of the query is polynomial. □

Remark 6.1 The probability measure used here giving equal weights to all inputs of a given size is not realistic in general. In the example of a single binary relation, it is equivalent to assuming probability 1/2 for each edge. Consequently, the relation will almost always contain about half of all pairs. Relations are usually more sparse than this.

For a more realistic approach we may adopt a model often used in the study of random graphs (see Bollabas [Bol85]). The idea is that rather than assuming probability 1/2 for each edge, we use a probability which is a function $p(n)$ of the input size n, which allows us to model some sparse graphs. Indeed, all of our calculations go through virtually unchanged for $p(n)$ such that for every positive ϵ and sufficiently large n,

$$n^{-\varepsilon} < p(n) < 1 - n^{-\varepsilon}.$$

For example, we could take $p(n) = n^{-1/\log\log n}$, which decreases to 0 as n increases. Spencer and Shelah [SS88] considered the problem of probabilities of first-order queries on random graphs when $p(n)$ is even smaller, but did not consider *while* queries or resource bounds. Again, this raises interesting questions with regard to our work.

7 Conclusion

By using a probabilistic approach to queries, we obtained surprising results about the important class of $while^+\!\downarrow$ queries. First, we showed that a powerful normal form for $while^+$ programs can be used to optimize $while^+$ computations, and that the expected benefit is high. We further showed that $while^+\!\downarrow$ programs are bounded almost everywhere, and that this can be used effectively to reduce the computation to a FO query in most cases. We considered the average complexity of *fixpoint* and *while*, and showed that in most cases these queries are much easier than the worst case analysis would suggest. The results use a powerful normal form for $while^+\!\downarrow$ queries, in conjunction with techniques related to 0-1 laws.

The results shown here may seem too good to be true. For instance, they show that *fixpoint* is on the average no harder than FO. In fact, every $while^+\!\downarrow$ program, which may involve computation of arbitrary complexity, reduces in most cases to just an FO query! To put things in perspective, it must be noted that the results are dependent on the fundamental assumption that no information is a priori known about the input, i.e., that *all inputs are equally likely.* Clearly, this is unrealistic in many applications. For instance, integrity constraints violate this assumption since some inputs are ruled out a priori. It may also be the case that the likelihood of the inputs is biased. For instance, an airline database consisting of edges representing links between cities is likely to yield a sparse graph; and in many graph applications, the out-degree is bounded.

As it turns out, the results obtained here generalize to some of the above situations. For instance, they carry through with certain sparsity assumptions on the input (see Remark 6.1). The results may extend to other situations as well, but this remains a question for future research.

References

[AG92] S. Abiteboul and A. Van Gelder. Optimizing active databases using the split technique. Technical report, INRIA, 1992.

[AS91] S. Abiteboul and E. Simon. Fundamental properties of deterministic and nondeterministic extensions of datalog. *Theoretical Computer Science*, 78:137–158, 1991.

[AV91a] S. Abiteboul and V. Vianu. Datalog extensions for database queries and updates. *Journal of Computer and System Sciences*, 43:62–124, 1991.

[AV91b] S. Abiteboul and V. Vianu. Generic computation and its complexity. In *Proc. ACM SIGACT Symp. on the Theory of Computing*, pages 209–219, 1991.

[AVV91] S. Abiteboul, M. Vardi, and V. Vianu. Fixpoint logics, relational machines, and computational complexity. Technical report, in preparation, 1991.

[AVV92] S. Abiteboul, M. Vardi, and V. Vianu. Computing with infinitary logic. Technical report, in preparation, 1992.

[Bol85] Béla Bollobás. *Random Graphs*. Academic Press, London, 1985.

[CH80] A.K. Chandra and D. Harel. Computable queries for relational data bases. *Journal of Computer and System Sciences*, 21(2):156–178, 1980.

[CH82] A.K. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25(1):99–128, 1982.

[Cha81] A.K. Chandra. Programming primitives for database languages. In *Proc. ACM Symp. on Principles of Programming Languages*, pages 50–62, 1981.

[Dah87] E. Dahlhaus. Skolem normal forms concerning the least fixpoint. In E. Borger, editor, *Computation Theory and Logic*, pages 101–106. Springer-Verlag LNCS 270, 1987.

[Fag90] R. Fagin. Finite-model theory - a personal perspective. In *Third Int'l. Conf. on Database Theory*, pages 3–24, 1990.

[Gel89] A. Van Gelder. The alternating fixpoint of logic programs with negation. In *Proc. ACM Symp. on Principles of Database Systems*, pages 1–11, 1989.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractibility: A Guide the to the Theory of NP-Completeness*. Freeman, New York, 1979.

[Gra83] E. Grandjean. Complexity of the first-order theory of almost all finite structures. *Inform. and Control*, 57:180–204, 1983.

[GS86] Y. Gurevich and S. Shelah. Fixed-point extensions of first-order logic. *Annals of Pure and Applied Logic*, 32:265–280, 1986.

[Gur91] Yuri Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42:346–398, 1991.

[Imm86] N. Immerman. Relational queries computable in polynomial time. *infcont*, 68:86–104, 1986.

[Imm87] N. Immerman. Expressibility as a complexity measure: Results and directions. Technical Report DCS-TR-538, Yale Univ., 1987.

[KA89] P. Kanellakis and S. Abiteboul. Deciding bounded recursion in database logic programs. *SIGACT News*, 20:4, 1989.

[Kan91] P. C. Kanellakis. Elements of relational database theory. In J. Van Leeuwen, editor, *Handbook of Theoretical Computer Science*, pages 1074–1156. North Holland, 1991.

[Kol91] P.G. Kolaitis. The expressive power of stratified logic programs. Technical report, to appear in Information and Computation, 1991.

[KP88] P.G. Kolaitis and C.H. Papadimitriou. Why not negation by fixpoint? In *Proc. ACM Symp. on Principles of Database Systems*, pages 231–239, 1988.

[KV87] Phokion G. Kolaitis and Moshe Y. Vardi. The decision problem for probabilities of higher order properties. In *Proc. 19th ACM Symp. on Theory of Computing*, pages 425–435, New York, 1987. Association for Computing Machinery.

[KV90a] P. Kolaitis and M. Vardi. 0-1 laws for infinitary logics. In *5th IEEE Symp. on Logic in Computer Science*, pages 156–167, 1990.

[KV90b] Phokion G. Kolaitis and Moshe Y. Vardi. 0-1 laws for infinitary logics. In *Proc. 5th IEEE Conf. on Logic in Computer Science*, pages 156–167, Los Angeles, 1990. IEEE Computer Society Press.

[KV92] P. Kolaitis and M. Vardi. 0-1 laws for infinitary logics. *Information and Computation*, to appear, 1992.

[Lev86] Leonard Levin. Average case complete problems. *SIAM Journal on Computing*, 15:285–286, 1986.

[SN91] S. Seshadri and J.F. Naughton. On the expected size of recursive Datalog queries. In *Proc. ACM Symp. on Principles of Database Systems*, pages 268–279, 1991.

[SS88] Saharon Shelah and Joel Spencer. Zero-one laws for sparse random graphs. *Amer. J. Math.*, 1:97–115, 1988.

[Ull88] J.D. Ullman. *Principles of Database and Knowledge Base Systems: Volume I and II.* Computer Science Press, 1988.

[Var82] M.Y. Vardi. The complexity of relational query languages. In *Proc. ACM SIGACT Symp. on the Theory of Computing*, pages 137–146, 1982.

[Var92] M. Vardi, 1992. email communication.

[Wil84] H.S. Wilf. Backtrack: an $o(1)$ expected time algorithm for the graph coloring problem. *Information Processing Letters*, 18:3:119–121, 1984.