Computability in Europe 2008

Logic and Theory of Algorithms

University of Athens
June 15-20, 2008

Title:

Query-Optimal Oracle Turing Machines for Type-2 Computations

Abstract:

With the notion of optimal programs defined in terms of Blum's complexity measure, the well known speed-up theorem states that there exist computable functions that do not have optimal programs. For type-2 computation, we argue that the same speed-up theorem can be obtained as long as the complexity measure satisfies Blum's complexity measure under the Oracle Turing Machine model. In the present paper, we consider the oracle query as a dynamic complexity measure. Since the query complexity is not a Blum's complexity measure, we have to characterize the notions of optimal programs in a very different way. We give three notions of query-optimal programs in different strength and argue that the stronger the notion of query-optimal programs we define, the more difficult to have query-optimal programs. In other words, with a stronger definition of query-optimal programs, one can prove an easier version of the speed-up theorem.

Keywords:

Type-2 Complexity Theory, Type-2 Computation, Query Optimal Programs, Speed-up Theorems

Author:

Chung-Chih Li, Ph.D.
Assistant Professor

School of Information Technology
P.O. Box 5150
Old Union
Illinois State University
Normal, IL 61790, USA

Tel: +1-309-438-7952
Fax: +1-309-438-5113

e-mail: cli2@ilstu.edu

Query-Optimal Oracle Turing Machines for Type-2 Computations

Chung-Chih Li

School of Information Technology
Illinois State University
Normal, IL 61790, USA

Abstract. With the notion of optimal programs defined in terms of Blum's complexity measure, the well known speed-up theorem states that there exist computable functions that do not have optimal programs. For type-2 computation, we argue that the same speed-up theorem can be obtained as long as the complexity measure satisfies Blum's complexity measure under the Oracle Turing Machine model. In the present paper, we consider the oracle query as a dynamic complexity measure. Since the query complexity is not a Blum's complexity measure, we have to characterize the notions of optimal programs in a very different way. We give three notions of query-optimal programs in different strength and argue that the stronger the notion of query-optimal programs we define, the more difficult to have query-optimal programs. In other words, with a stronger definition of query-optimal programs, one can prove an easier version of the speed-up theorem.

1 Introduction

Searching for the best algorithm to solve a concerned computable problem is a primary task for computer scientists. However, the speed-up theorem tells us that, theoretically, we cannot succeed on all problems. In fact, the speed-up phenomena have been extensively studied by mathematicians for more than a half century, which in logical form was first remarked by Gödel in terms of the length of theorem proving [5, 6, 13]. Blum [1, 2] re-discovered the speed-up phenomenon in the context of computational complexity.

Blum's speed-up theorem is a result of his axiomatization of complexity measures. Fix a programming system for Turing machines and let φ_i denote the function computed by the i^{th} Turing machine and Φ_i denote the cost function associated to it. Blum considers that dynamic complexity measures should meet the following two requirements: for any $i, x, m \in \mathbf{N}$, (i) φ_i is convergent on x (denoted by $\varphi_i(x)\downarrow$) if and only if Φ_i is convergent on x (denoted by $\Phi_i(x)\downarrow$) and (ii) $\Phi_i(x) = m$ is effectively decidable. The two axioms have successfully lifted the study of complexity theory to an abstract level with rich results that are independent from any specific machine models. At type-1, the meaning of "more efficient programs" is intuitive: For a computable function f, we say that program j is more efficient than program i if $\varphi_i = \varphi_j = f$ and for all but finitely

many x we have $\Phi_j(x) < \Phi_i(x)$. Precisely, Blum [1, 2] formulates the speed-up theorem as follows: For any recursive function r, there exists a recursive function f such that

$$(\forall\, i : \varphi_i = f)\ (\exists j : \varphi_j = f)\ (\overset{\infty}{\forall}\, x)\ \left[r(\Phi_j(x)) \leq \Phi_i(x)\right]. \tag{1}$$

There are many sources for detailed proofs and discussions regarding this curious speed-up phenomenon, e.g., [1–4, 6, 11–14]. We shall skip all irrelevant details due to the space constraints.

When we try to lift the complexity theory to type-2 computation, we want to know whether or not the same speed-up phenomenon can be described in terms of some complexity measures that are sensible in the context of type-2 computation. Clearly, time and space remain meaningful in type-2 computation. Since time and space are Blum's complexity measures, we have not faced too many difficulties in translating the original proofs of the speed-up theorem and its variants into type-2 (see [9]). For query complexity, on the other hand, the two Blum's axioms fail. As a result, we can't reuse the same concepts and techniques but develop our new ones in order to investigate this specular speed-up phenomenon.

2 Type-2 Computation Using Oracle Turing Machines

In this section we will provide necessary notations for type-2 computation; detailed discussion about the type-2 complexity theory can be found in [7–10]. We consider natural numbers as type-0 objects and functions over natural numbers as type-1 objects. For type-2 objects, they are *functionals* that take as inputs and produce as outputs type-0 or type-1 objects. We consider objects of lower type as special cases of higher type, i.e., type-0 $\subset$ type-1 $\subset$ type-2. Without loss of generality we restrict our type-2 functionals to the standard type $\mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$, where $\mathcal{T}$ is the set of total functions and $\rightharpoonup$ means possibly partial.

We use the Oracle Turing Machine (OTM) as our standard computing formalism for type-2 computation. An OTM is a Turing machine equipped with a function oracle. We say that a type-2 functional $F : \mathcal{T} \times \mathbf{N} \rightharpoonup \mathbf{N}$ is computable if there is an OTM that computes F as follows. Before the OTM begins to run, the type-1 argument should be presented to the OTM as an oracle. Every OTM has two extra tapes called query-tape and answer-tape for oracle queries and oracle answers, respectively. During the course of the computation, the OTM may enter a special state called query-state where the oracle will read the query left on the query-tape and return its answer to the answer-tape. We can fix a programming system $\langle \widehat{\varphi}_i \rangle_{i \in \mathbf{N}}$ for OTM's and let $\widehat{M}_e$ denote the OTM with index e that computes $\widehat{\varphi}_e$.

Let $\mathcal{F}$ denote the set of finite functions over natural numbers, i.e., $\sigma \in \mathcal{F}$ iff the domain of σ is a subset of $\mathbf{N}$ and its cardinality is in $\mathbf{N}$. Given $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, $F(f, x) \downarrow = y$ denotes that F is defined at (f, x) and its value is y. Since if F is computable, F must be *continuous* (i.e., *compact* and *monotone*), it follows that if $F(f, x) \downarrow = y$, there must be $\sigma \in \mathcal{F}$ with $\sigma \subset f$ such that for every extension

η of σ (i.e., $\sigma \subseteq \eta$), we have $F(f,x) \downarrow= F(\sigma,x) \downarrow= F(\eta,x) \downarrow= y$. We say that (σ,x) and (η,x) are locking fragments of F on (f,x). For any $\sigma \in \mathcal{F}$, let $(\!(\sigma)\!)$ denote the set of total extensions of σ, i.e., $(\!(\sigma)\!) = \{f \in \mathcal{T} \mid \sigma \subset f\}$. Also, if $(\sigma,x) \in \mathcal{F} \times \mathbf{N}$, let $(\!(\sigma,x)\!) = \{(f,x) \mid f \in (\!(\sigma)\!)\}$. If σ_1 and σ_2 are consistent, we have $(\!(\sigma_1)\!) \cap (\!(\sigma_2)\!) = (\!(\sigma_1 \cup \sigma_2)\!)$; otherwise, $(\!(\sigma_1)\!) \cap (\!(\sigma_2)\!) = \emptyset$. The union operation $(\!(\sigma_1)\!) \cup (\!(\sigma_2)\!)$ is conventional.

3 Query-optimal Programs

Only in type-2 computation can we use the number of queries made throughout the course of the computation as a dynamic complexity measure. We are interested in whether or not there is a *query-optimal program* for any given type-2 computable functional; if there is not, can we have a speed-up theorem in terms of query-complexity. It turns out that the query-complexity fails to satisfy Blum's two axioms. Consequently, the arguments in the original proof of the speed-up theorem are no longer valid. We thus need a new approach to understand this curious phenomenon. In this section we define a few alternative notions of query-optimal programs. These notions must be sensible and intuitive.

3.1 Unnecessary Queries

We shall argue that some intuitive notions of query-optimal programs are not flexible enough to derive interesting results. We say that an oracle query is necessary if its answer returned from the oracle will affect the result of the computation. Intuitively, a query-optimal program should be a program that does not make any unnecessary queries. Unfortunately, this notion of query-optimal programs is too strong. It is easy to argue that there are functionals that always make unnecessary oracle queries. Consider the following functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ defined by,

$$F(f,x) = \begin{cases} f(0)+1 & \text{if } \varphi_x(x) \downarrow \text{ in } f(0) \text{ steps;} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

Clearly, F is computable and total. Fix any a such that φ_a is diverged on a (denoted by $\varphi_a(a) \uparrow$). Then, on input (f,a), the value of $f(0)$ only affects the speed of computing $F(f,a)$. Thus, $F(f,a) = 0$ for any $f \in \mathcal{T}$, and hence $(\emptyset,a)$ is the minimal locking fragment of F on (f,a). That means any queries made during the computation of F on (f,a) are unnecessary. By contradiction, if there were an OTM that would not make any unnecessary queries for F, one could modify such OTM to solve the halting problem, which is impossible. Therefore, the type-2 functional defined in (2) can't be computed by an OTM without making unnecessary queries. From this example, it is clear that to require query-optimal programs to be ones that do not make unnecessary queries is too restricted. We will loose the requirements by allowing a small amount of unnecessary queries. By "a small amount", we means "a compact set" in some topology determined by the concerned computation. We formalize our idea in the next subsection.

3.2 Unnecessary Queries in Compact Sets

We use $Q(s, e, f, x)$ to denote the collection of queries made during the run time of OTM $\widehat{M}_e$ on (f, x) up to s steps. Formally,

Definition 1. *Let e be a $\widehat{\varphi}$-program. Define $Q : (\mathbf{N} \times \mathbf{N} \times \mathcal{T} \times \mathbf{N}) \to \mathcal{F}$ by: $Q(s, e, f, x) = \tau$, where $\tau \in \mathcal{F}$ with $\tau \subset f$ and $\mathsf{dom}(\tau)$ is the collection of queries made during the course of the computation of OTM $\widehat{M}_e$ on (f, x) in s steps.* ■

Clearly, $Q(s, e, f, x)$ is monotone in s, i.e., $s \in \mathbf{N}$, $Q(s, e, f, x) \subseteq Q(s+1, e, f, x)$.

Definition 2. *Let $e, x \in \mathbf{N}$ and $f \in \mathcal{T}$. We say that $Q(\cdot, e, f, x)$ is defined in the limit if and only if there exist $s \in \mathbf{N}$ and $\tau \in \mathcal{F}$ such that, for every $s' \geq s$,*

$$Q(s, e, f, x) = Q(s', e, f, x) = \tau.$$

In the case above, we write $\lim\limits_{s\to\infty} Q(s, e, f, x) \downarrow = \tau$. ■

We wish to treat the set of queries an OTM made as a dynamic complexity measure, but it fails to satisfy Blum's two axioms for complexity measure. That is, $\lim\limits_{s\to\infty} Q(s, e, f, x) \downarrow$ does not mean that $\widehat{\varphi}_e(f, x) \downarrow$, and hence Blum's first axiom fails. Moreover, for some $\sigma \in \mathcal{F}$ with $\sigma \subset f$, $\lim\limits_{s\to\infty} Q(s, e, f, x) = \sigma$ is not necessarily decidable, and hence Blum's second axiom fails too.

Let $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be computable. Since we mean to characterize some $\widehat{\varphi}$-programs $e_1, e_2, \ldots, e_n$ that compute the same functional F in terms of queries, the topology $\mathbb{T}(\cdots)$ defined in [7–10] cannot serve as the relative topology in which we need a workable notion of compactness. This is because $\widehat{\varphi}_{e_1} = \widehat{\varphi}_{e_2} = \cdots \widehat{\varphi}_{e_n} = F$ and thus $\mathbb{T}(\widehat{\varphi}_{e_1}, \widehat{\varphi}_{e_2}, \ldots, \widehat{\varphi}_{e_n}) = \mathbb{T}(F)$, and hence the complexity of unnecessary queries cannot be described by the compact sets in $\mathbb{T}(F)$. The product topology, $\mathbb{T} \times \mathbb{N}$ (Baire and the Discrete topologies), is fine enough to describe the needed compact sets but it suffers the same problem as discussed in [7–10]. We thus introduce another family of topologies as follows.

Definition 3. *Given $e_1, e_2, \ldots, e_n \in \mathbf{N}$, let $B(e_1, e_2, \ldots, e_n) \subseteq \mathcal{T} \times \mathbf{N}$ be defined as follows: $(\sigma, a) \in B(e_1, e_2, \ldots, e_n)$ if and only if there exists some $f \in \mathcal{T}$ such that, for each $i \in \{1, \ldots, n\}$, we have*

$$\lim_{s\to\infty} Q(s, e_i, f, a) \downarrow = \sigma_i,$$

and $\sigma = \bigcap\limits_{i\in\{1\ldots n\}} \sigma_i$. Let $\mathbb{Q}(e_1, \ldots, e_n)$ be the topology on $\mathcal{T} \times \mathbf{N}$ defined by the basic open sets in $\mathcal{B}$, where

$$\mathcal{B} = \left\{ (\!(\sigma, x)\!) \;\middle|\; (\sigma, x) \in B \right\}.$$

■

Since a $\widehat{\varphi}$-program e may make unnecessary queries outside the domains of its minimal locking fragments, it follows that the topology $\mathbb{Q}(e)$ may be finer than the topology $\mathbb{T}(\widehat{\varphi}_e)$ which is defined by the minimal locking fragments of $\widehat{\varphi}_e$. That is, $\mathbb{T}(\widehat{\varphi}_e) \subseteq \mathbb{Q}(e)$. In other words, if e is a $\widehat{\varphi}$-program for $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$, then $\mathbb{T}(F) \subseteq \mathbb{Q}(e)$. On the other hand, if $\mathbb{Q}(e) \subset \mathbb{T}(F)$, then we can conclude that e cannot be a $\widehat{\varphi}$-program for F. In general, we have

$$\mathbb{T}(\widehat{\varphi}_i, \widehat{\varphi}_j) \subseteq \mathbb{Q}(i, j).$$

For convenience, we define the following notations. Let $\mathsf{card}(S)$ be the cardinality of set S. Suppose i and j are two $\widehat{\varphi}$-programs that compute the same functional, i.e., $\widehat{\varphi}_i = \widehat{\varphi}_j$. Define

$$\begin{aligned}
Q_{[i<j]} &= \left\{(f,x) \mid \mathsf{card}(\lim_{s\to\infty} Q(s,i,f,x)) < \mathsf{card}(\lim_{s\to\infty} Q(s,j,f,x))\right\},\\
Q_{[i=j]} &= \left\{(f,x) \mid \mathsf{card}(\lim_{s\to\infty} Q(s,i,f,x)) = \mathsf{card}(\lim_{s\to\infty} Q(s,j,f,x))\right\},\\
Q_{[i>j]} &= \left\{(f,x) \mid \mathsf{card}(\lim_{s\to\infty} Q(s,i,f,x)) > \mathsf{card}(\lim_{s\to\infty} Q(s,j,f,x))\right\}.
\end{aligned}$$

The following definition makes use of the compactness in the topology we just mentioned to define the meaning of "better" $\widehat{\varphi}$-programs in terms of query-complexity.

Definition 4 (Query Speed-up). *Let $\widehat{\varphi}_i = \widehat{\varphi}_j$.*

1. *We say that j is a **weakly** query sped-up version of i if and only if*

$$Q_{[i<j]} \text{ is compact and } Q_{[i>j]} \text{ is noncompact in } \mathbb{Q}(i,j).$$

   *If j is a **weakly** query sped-up version of i, we write $\widehat{\varphi}_i \prec^*_Q \widehat{\varphi}_j$.*
2. *We say that j is a **strongly** query sped-up version of i if and only if*

$$Q_{[i<j]} \cup Q_{[i=j]} \text{ is compact in } \mathbb{Q}(i,j).$$

   *If j is a **strongly** query sped-up version of i, we write $\widehat{\varphi}_i \prec\!\prec^*_Q \widehat{\varphi}_j$.* ■

Suppose that we are to write a new $\widehat{\varphi}$-program j for $\widehat{\varphi}_i$. Intuitively, if the new $\widehat{\varphi}$-program j can improve the query-complexity on a "large" (noncompact) set of inputs and leaves a "small" (compact) set of inputs on which j queries more than i does, we say that j is a *weakly* query sped-up version of i. Note that, we do not care how many inputs on which the query-complexity remains unchanged. Since $\mathbb{T}(\widehat{\varphi}_j) \subseteq \mathbb{Q}(i,j)$, it follows that, if $Q_{[i<j]}$ is compact in $\mathbb{Q}(i,j)$, so is it in $\mathbb{T}(\widehat{\varphi}_j)$, and hence we can patch $\widehat{\varphi}$-program j on $Q_{[i<j]}$ so that the patched $\widehat{\varphi}$-program will never query more than i does. Whereas, if $\widehat{\varphi}$-program j is a *strongly* query sped-up version of i, then the set on which j does not improve its query-complexity must be "small" (compact). Note that, if $Q_{[i<j]} \cup Q_{[i=j]}$ is compact in $\mathbb{Q}(i,j)$, then $Q_{[i>j]}$ must be noncompact in $\mathbb{Q}(i,j)$. Clearly, for $\widehat{\varphi}_i = \widehat{\varphi}_j$, we have the following implication:

$$\widehat{\varphi}_i \prec\!\prec^*_Q \widehat{\varphi}_j \implies \widehat{\varphi}_i \prec^*_Q \widehat{\varphi}_j.$$

Definition 5 (Query-optimal $\widehat{\varphi}$-programs). *Let $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ be a computable functional. Let e be a $\widehat{\varphi}$-program for F.*

1. *We say that e is an **absolute** query-optimal $\widehat{\varphi}$-program for F if and only if, on every $(f, x) \in \mathcal{T} \times \mathbf{N}$, the OTM $\widehat{M}_e$ does not make unnecessary queries during the course of computing $\widehat{\varphi}_e(f, x)$.*
2. *We say that e is a **strong** query-optimal $\widehat{\varphi}$-program for F if and only if there is no $\widehat{\varphi}$-program i for F such that, $\widehat{\varphi}_e \prec^*_Q \widehat{\varphi}_i$.*
3. *We say that e is a **weak** query-optimal $\widehat{\varphi}$-program for F if and only if there is no $\widehat{\varphi}$-program i for F such that, $\widehat{\varphi}_e \prec\!\prec^*_Q \widehat{\varphi}_i$.* ■

Definition 5 gives us three versions of query-optimal $\widehat{\varphi}$-programs, where version 1 is the strongest notion and version 3 is the weakest one. Suppose that e is a $\widehat{\varphi}$-program for $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$. The following implications are straightforward:

1. If e is an absolute query-optimal $\widehat{\varphi}$-program for F, then it is also a strong query-optimal $\widehat{\varphi}$-program for F.
2. If e is an strong query-optimal $\widehat{\varphi}$-program for F, then it is also a weak query-optimal $\widehat{\varphi}$-program for F.

Moreover, if $\mathbb{T}(\widehat{\varphi}_e) = \mathbb{Q}(e)$, then e is an absolute query-optimal $\widehat{\varphi}$-program for $\widehat{\varphi}_e$. We shall argue that the stronger the notion of *query-optimum*, the easier we can obtain a speed-up theorem. In other words, it is more difficult to obtain a query-optimal OTM when the notion of query-optimum becomes stronger. However, we do not know if the speed-up phenomenon still exists under our weakest notion of query-optimum; neither do we know how far can one weaken the notion of query-optimal programs to do away the speed-up phenomenon while keeping the notion nontrivial. We give partial results in the next section.

4 Query-optimal Programs

With a fixed complexity measure that satisfies Blum's axioms, the original speed-up theorem states that there is a total computable function without optimal program for it. Although query-complexity does not satisfy the Blum's axioms, each of the three versions of query-optimal programs gives rise to the following questions:

1. Does every computable functional $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ always have a query-optimal $\widehat{\varphi}$-program for it?
2. If the answer to the first question is negative, then can we construct the query-speed-able functional?
3. If a given $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ does have a query-optimal $\widehat{\varphi}$-programs for it, then can we uniformly construct a query-optimal program for F from an arbitrary $\widehat{\varphi}$-program for F?
4. Suppose there is no query-optimal $\widehat{\varphi}$-program for $\widehat{\varphi}_{e_0} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$. Can we effectively construct an infinite sequence of query-sped-up version of $\widehat{\varphi}$-programs from e_0? That is, can we have $e_0, e_1, e_2, \ldots$ such that, for each $i \in \mathbf{N}$, e_{i+1} is a query-sped-up version of e_i?

We have a positive answer to the second question with respect to the *absolute* and *strong* versions of query-optimal programs. Thus, a negative answer to the first question follows immediately. Compared to the original *nonconstructive* speed-up theorem, the speedable functional for the second question is very simple. However, we do not know if there is a computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ that can forever strongly speed-up, i.e., we do not know if there is a computable $F : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ without a weak query-optimal $\widehat{\varphi}$-program. The answer to the third question is negative given by Theorem 4. The fourth question is still open. That is, given $\widehat{\varphi}_{e_0} : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ that is known to be speedable, we do not know if there is an effective way to construct a sequence of $\widehat{\varphi}$-programs such that,

$$\widehat{\varphi}_{e_0} \prec_Q^* \widehat{\varphi}_{e_1} \prec_Q^* \widehat{\varphi}_{e_2} \prec_Q^* \cdots\cdots$$

or similarly,

$$\widehat{\varphi}_{e_0} \prec\prec_Q^* \widehat{\varphi}_{e_1} \prec\prec_Q^* \widehat{\varphi}_{e_2} \prec\prec_Q^* \cdots\cdots .$$

Let $\varphi_e(x) \downarrow_s$ denote the case that the Turing Machine M_e, on x, halts in s steps. We now define a useful functional $K : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ in the following.

Definition 6. *Define* $K : \mathcal{T} \times \mathbf{N} \to \mathbf{N}$ *by*

$$K(f,x) = \begin{cases} \varphi_x(x)+1 & \text{if } \varphi_x(x)\downarrow_s, \text{ where } s = \sum_{i=0}^{f(0)} f(i); \\ 0 & \text{otherwise.} \end{cases}$$

■

Let $\mathbf{H}$ denote the set $\{x|\varphi_x(x)\downarrow\}$. For any $(f,x) \in \mathcal{T} \times \mathbf{N}$, if $x \notin \mathbf{H}$, then any $\widehat{\varphi}$-program that computes K, on input (f,x), may ask arbitrarily many unnecessary queries, where the number of queries depends on the value of $f(0)$. It is easy to prove that there is no $\widehat{\varphi}$-program for K with all unnecessary queries removed (Theorem 1). Even if we lower the standard to strong query-optimal programs, we still cannot have a query-optimal $\widehat{\varphi}$-program for K (Theorem 2). Theorem 3 states that K does have a weak query-optimal program for it.

Theorem 1. *K has no absolute query-optimal $\widehat{\varphi}$-program.*

Proof: We show that from an absolute query-optimal $\widehat{\varphi}$-program for K, if any, one can construct a solution to the halting problem.

Suppose, by contradiction, $\widehat{\varphi}_e$ is an absolute query-optimal $\widehat{\varphi}$-program for K. Clearly, if $x \notin \mathbf{H}$, then for any $f \in \mathcal{T}$, $K(f,x) = 0$. On the other hand, if $x \in \mathbf{H}$, there must be two distinct $f, g \in \mathcal{T}$ such that $\widehat{\varphi}_e(f,x) \neq \widehat{\varphi}_e(g,x)$. Thus, some queries to the oracles f and g must be made. Therefore, given any $x \in \mathbf{N}$, we can run $\widehat{\varphi}_e(\emptyset^{\sim 0}, x)$ first, where $\emptyset^{\sim 0}$ is the zero everywhere function. Then, if no queries were made during the course of computation, we know that $x \notin \mathbf{H}$. Hence, $\mathbf{N} - \mathbf{H}$ is recursively enumerable, a contradiction. □

Theorem 2. *K has no strong query-optimal $\widehat{\varphi}$-program.*

Proof: Suppose, by contradiction, $\widehat{\varphi}_e$ is a strong query-optimal $\widehat{\varphi}$-program for K. Let $i \notin \mathbf{H}$ such that, for any $f \in \mathcal{T}$, the OTM, $\widehat{M}_e$, on (f, i), will make oracle queries at points $0, 1, 2, \ldots, f(0)$. Such i must exist, otherwise the halting problem can be solved by a similar argument given for the proof of Theorem 1. We construct a *weakly* sped-up version of e as shown in Figure 1.

```
Program
        input f : 𝒯, x : N;
        if x = i and f(0) is even
            then output 0;
            else output φ̂_e(f, x);    /* by running φ̂-program e on (f, x) */
End program
```

Fig. 1. A Weakly Sped-up Version of $\widehat{\varphi}_e$

Let the index of the $\widehat{\varphi}$-program in Figure 1 be p. Clearly, if $x \neq i$ or if $f(0)$ is odd, then the two $\widehat{\varphi}$-programs, e and p, will perform the same computation, and hence the two make the same oracle queries. The two computations differ on the following set:

$$\left\{(f, i) \;\middle|\; f(0) \text{ is even}\right\}. \tag{3}$$

Clearly, the set above is noncompact in $\mathbb{Q}(e, p)$. According to the program p and the assumption on i, the following set is the same as (3),

$$\left\{(f, i) \;\middle|\; \mathsf{card}(\lim_{s\to\infty} Q(s, e, f, i)) > \mathsf{card}(\lim_{s\to\infty} Q(s, p, f, i))\right\}.$$

Thus, $Q_{[e>p]}$ is noncompact in $\mathbb{Q}(e, p)$. Moreover, $Q_{[e<p]} = \emptyset$, and hence $Q_{[e<p]}$ is compact in $\mathbb{Q}(e, p)$. It follows that p is a weakly sped-up version of e, i.e., $\widehat{\varphi}_e \prec^*_Q \widehat{\varphi}_p$. Therefore, e cannot be a strong query-optimal $\widehat{\varphi}$-program for K. □

Theorem 3. *There is a weak query-optimal $\widehat{\varphi}$-program for K.*

Sketch of Proof: It is impossible to strongly speed-up any $\widehat{\varphi}$ forever. Otherwise, we can reach two $\widehat{\varphi}$ programs e and p such that, $\widehat{\varphi}_e = \widehat{\varphi}_p = K$ and $\widehat{\varphi}_e \prec\!\prec^*_Q \widehat{\varphi}_p$. Then, we can compare the queries made during the course of the computations between $\widehat{\varphi}_e$ and $\widehat{\varphi}_p$ to solve the halting problem. Therefore, a weak query-optimal $\widehat{\varphi}$-program for K must exist. We skip details of the proof. □

However, we do not know if there is a computable type-2 functional such that, there is no weak query-optimal $\widehat{\varphi}$-program for it. In other words, we do not if there is a speed-up phenomenon associated with the notion of weak query-optimum.

Suppose we know that a query-optimal program for $\widehat{\varphi}_e$ does exist. In general, there is no effective way to construct a query-optimal version of e in any strength, i.e., absolute, strong, and weak query-optimum. We state this in the following theorem, which is the strongest version in the sense that the weakest version of query-optimal programs is considered.

Theorem 4. *There is no recursive $r : \mathbf{N} \to \mathbf{N}$ such that, for every $\widehat{\varphi}$-program e, if $\widehat{\varphi}_e$ is total and there is a weak query-optimal program for it, $r(e)$ is a weak query-optimal $\widehat{\varphi}$-program for $\widehat{\varphi}_e$.*

Proof: By contradiction, suppose such recursive $r : \mathbf{N} \to \mathbf{N}$ exists. Consider the following program shown in Figure 2. Suppose that, by the recursion theorem, the index of the $\widehat{\varphi}$-program is e.

```
Program e
        input f : 𝒯, x : N;
        Q ⟵ Q(f(0), r(e), f, x);
        if φ̂_{r(e)}(f, x) converges in f(0) steps
            then output f(max(dom(Q)) + 1);
            else output f(1);
End program e
```

Fig. 2. A $\widehat{\varphi}$-program with index e for Theorem 4

According to the construction, it is clear that e is a total $\widehat{\varphi}$-program. By assumption, $r(e)$ is a weak query-optimal version of e. On any $(f, x) \in \mathcal{T} \times \mathbf{N}$, the OTM $\widehat{M}_{r(e)}$ will never halt in $f(0)$ steps, otherwise the OTM $\widehat{M}_{r(e)}$ does not compute $\widehat{\varphi}_e$. This is because if the OTM $\widehat{M}_{r(e)}$ on (f, x) halts in $f(0)$ steps, Q must contain the complete collection of queries made by the OTM $\widehat{M}_{r(e)}$ on (f, x). But $\widehat{\varphi}_e(f, x)$ is the value of f at a point not in Q.

Thus, we conclude that for every $(f, x) \in \mathcal{T} \times \mathbf{N}$, $\widehat{\varphi}_e(f, x) = f(1)$. But if that is the case, $f(0)$ will not be queried by the OTM $\widehat{M}_{r(e)}$, namely, the value of $f(0)$ does not affect the computation of $\widehat{M}_{r(e)}$ at all. Thus, if $f(0)$ is sufficiently large, $\widehat{\varphi}_{r(e)}(f, x)$ will converge in $f(0)$ steps, and hence $\widehat{\varphi}_e(f, x) = f(\mathsf{max}(\mathsf{dom}(Q))+1)$. Thus, the OTM $\widehat{M}_{r(e)}$ does not compute $\widehat{\varphi}_e$.

Therefore, $r(e)$ cannot be a weak query-optimal version of $\widehat{\varphi}$-program e. □

5 Conclusion

The oracle query is a unique dynamic complexity measure in type-2 computation. Although Blum's complexity measure has created a rich chapter in machine independent complexity theory, it is not appropriate to impose the same requirement to query complexity. We therefore provide new notions in order to describe the concept of query-optimal programs in type-2 computation. We obtain a speed-up theorem under a reasonable notion of query-optimal programs, which means that there exists a computable type-2 functional that does not have a query optimal OTM for it. Our version of speed-up theorem is stronger than the classical one in a sense that the speedable functional, K, does not depend on the speedup factor, i.e., the computable function r in equation (1).

Clearly, there are many open questions that deserve further invitation. For example, can we uniformly construct a query-sped-up program for our functional K? Do we have a speed-up theorem under our weaker notions of query-optimal programs? Some classical questions can also be asked. For examples, do we have a union theorem? gap theorem? compression theorem? Or do we have a complexity hierarchy characterized by the query-complexity? Since Blum's two axioms cannot be applied to the query-complexity, the approach and results must be very different from the classical ones. It seems to us that the framework proposed in this paper may be a feasible direction for computer science theorists to study the query complexity closely.

References

1. Manuel Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM*, 14(2):322–336, 1967.
2. Manuel Blum. On effective procedures for speeding up algorithms. *Journal of the ACM*, 18(2):290–305, 1971.
3. Critian Calude. *Theories of Computational Complexity.* Number 35 in Annals of Discrete Mathematics. North-Holland, Elsevier Science Publisher, B.V., 1988.
4. Nigel Cutland. *Computability: An introduction to recursive function theory.* Cambridge, New York, 1980.
5. Martin Davis, editor. *The Undecidable.* Raven Press, New York, 1965.
6. Kurt Gödel. Über die länge der beweise. *Ergebnisse eines Math. Kolloquiums*, 7:23–24, 1936. Translation in [5], pages 82-83, "On the length of proofs.".
7. Chung-Chih Li. Asymptotic behaviors of type-2 algorithms and induced Baire topologies. In *Proceedings of the Third International Conference on Theoretical Computer Science*, pages 471–484, Toulouse, France, August 2004.
8. Chung-Chih Li. Clocking type-2 computation in the unit cost model. In Arnold Beckmann, Ulrich Berger, Benedikt Löwe, and John V. Tucker, editors, *Proceedings of Computability in Europe, CiE 2006: Logical Approaches to Computational Barriers, CSR 7-2006*, pages 182–192, Swansea, UK, June/July 2006.
9. Chung-Chih Li. Speed-up theorems in type-2 computation. In S. Barry Cooper, Benedikt Löwe, and Andrea Sorbi, editors, *Proceedings of Computability in Europe, CiE 2007: Computation and Logic in the Real World*, pages 478–487, Siena, Italy, June 2007. Springer, LNCS 4497.
10. Chung-Chih Li and James S. Royer. On type-2 complexity classes: Preliminary report. In *Proceedings of the Third International Workshop on Implicit Computational Complexity*, pages 123–138, Aarhus, Denmark, May 2001.
11. A. R. Meyer and P. C. Fischer. Computational speed-up by effective operators. *The Journal of Symbolic Logic*, 37:55–68, 1972.
12. Joel I. Seiferas. Machine-independent complexity theory. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 163–186. North-Holland, Elsevier Science Publisher, B.V., 1990. MIT press for paperback edition.
13. P. Van Emde Boas. Ten years of speed-up. *Proceedings of the Fourth Symposium Mathematical Foundations of Computer Science 1975*, pages 13–29, 1975. Lecture Notes in Computer Science.
14. Klaus Wagner and Gerd Wechsung. *Computational Complexity.* Mathematics and its applications. D. Reidel Publishing Company, Dordrecht, 1985.