On the Time Complexity of Computer Viruses

Zhi-hong Zuo, Qing-xin Zhu, and Ming-tian Zhou, *Member, IEEE*

***Abstract*—Computer viruses can disable computer systems not only by destroying data or modifying a system's configuration, but also by consuming most of the computing resources such as CPU time and storage. The latter effects are related to the computational complexity of computer viruses. In this correspondence, we investigate some issues concerning the time complexity of computer viruses, and prove some known experimental results mathematically. We prove that there exist computer viruses with arbitrarily long running time, not only in the infecting procedure but in the executing procedure. Moreover, we prove that there are computer viruses with arbitrarily large time complexity in the detecting procedure, and there are undecidable computer viruses that have no "minimal" detecting procedure.**

***Index Terms*—Computational complexity, computer viruses, detection, infection, time complexity.**

I. Introduction

The first abstract theory of computer viruses is the viral set theory given by Cohen, based on the Turing machine [1], [2]. A viral set is defined by (M, V) where M is a Turing machine and V is a nonempty set of programs on the Turing machine. Each $v \in V$ is called a computer virus and satisfies the following condition: if it is contained in the tape at time t, then there exist a time t' and a $v' \in V$ such that v' is not contained in the tape at time t, but contained in the tape at time t'. The most important one of Cohen's theorems is about the undecidability of computer viruses [1], [2].

In a different approach, Adleman developed an abstract theory of computer viruses based on recursive functions [3]. In his definition, a virus is a total recursive function v which applies to all programs x (the Gödel numberings of programs), such that $v(x)$ has characteristic behaviors of computer viruses such as *injury*, *infection*, and *imitation*. Furthermore, Adleman proved that the set of computer viruses is $\prod_2$-complete [3].

Although several abstract theories about computer viruses were established and many important results were derived from these theories, we know little about the computational complexity of computer viruses. Very little research has been done so far in this respect. Adleman [3] discussed some problems but failed to give any conclusion. Spinellis [4] proved that to reliably identify the bounded-length of computer viruses is NP-complete. To our best knowledge, there is no more results on computational complexity of computer viruses.

In the following, we investigate some issues on the time complexity of computer viruses. We focus on two kinds of time complexity: the time complexity of computer viruses in their computational environment, and the time complexity of detecting computer viruses. It is well known that there exist arbitrarily complex recursive functions [5], [6], but we cannot apply this fact to computer viruses directly, because computer viruses are some "fixpoints" of recursive operations [3]. On the other hand, theoretical results about the time complexity of detecting computer viruses are very important to antivirus practice.

Manuscript received July 13, 2004; revised December 26, 2004.

The authors are with the College of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, Sichuan, 610054 China (e-mail: zhzuo@uestc.edu.cn;qxzhu@uestc.edu.cn; mtzhou@uestc.edu.cn).

Communicated by T. Johansson, Associate Editor for Complexity and Cryptography.

Digital Object Identifier 10.1109/TIT.2005.851780

The structure of the correspondence is as follows: In Section II, we introduce some notations; in Section III, we give definitions of computer viruses based on recursive functions. In Section IV, we prove some auxiliary lemmas and then derive main results about time complexity of computer viruses. In Section V, we give a brief summary and some suggestions for future research.

II. Preliminaries

We describe some notations below.

Let $\mathbb{N}$ be the set of all natural numbers and S be the set of all finite sequences of natural numbers. For $s_1, s_2, \ldots, s_n \in S$, let $\langle s_1, s_2, \ldots, s_n \rangle$ denote a computable injective function from S^n to $\mathbb{N}$ and its inverse is also computable. If $f : \mathbb{N} \to \mathbb{N}$ is a partial function, for $s_1, s_2, \ldots, s_n \in S$, we write $f(s_1, s_2, \ldots, s_n)$ instead of $f(\langle s_1, s_2, \ldots, s_n \rangle)$. Similarly, for $i_1, i_2, \ldots, i_n \in \mathbb{N}$, let $\langle i_1, i_2, \ldots, i_n \rangle$ denote a computable injective function from $\mathbb{N}^n$ to $\mathbb{N}$, satisfying $\langle i_1, i_2, \ldots, i_n \rangle \geq i_m$ for all $1 \leq m \leq n$, and its inverse is also computable. For a partial function $f : \mathbb{N}^n \to \mathbb{N}$, let $f(i_1, i_2, \ldots, i_n)$ represent $f(\langle i_1, i_2, \ldots, i_n \rangle)$.

For a sequence $p = (i_1, i_2, \ldots, i_k, \ldots, i_n) \in S$, let $(p)_k$ denote its kth element, and $p[j_k/i_k]$ denote the sequence obtained by replacing i_k with j_k in p, that is,

$$p[j_k/i_k] = (i_1, i_2, \ldots, j_k, \ldots, i_n).$$

If v is a computable function, $p[v(i_k)/i_k]$ is simply written as $p[v(\underline{i_k})]$. If more than one elements in p are replaced or evaluated by some computable functions, we write the result as

$$p[j_{k_1}/i_{k_1}, j_{k_2}/i_{k_2}, \ldots, j_{k_l}/i_{k_l}] \text{ or } p[v_1(\underline{i_{k_1}}), v_2(\underline{i_{k_2}}), \ldots, v_l(\underline{i_{k_l}})]$$

respectively.

Adopting Adleman's notations [3], let $\phi_P(d, p)$ denote a function computed by a computer program P in the running environment (d, p) where d represents *data* (including clock, spaces of diskettes, and so on) and p represents *programs* (including operating systems) stored on computers. If the index (the Gödel numbering) of P is e, the function is also denoted by $\phi_e(d, p)$. The domain and range are denoted by W_e and E_e, respectively. If h is a recursive function, we also use the symbols W_h and E_h for its domain and range. It is worth noting that there is no essential distinction between d and p, as in the case of Von Neumann machine. In this correspondence, we use the symbol (d, p) just for easy understanding.

For any program P, let t_P be the function defined as follows:

$$\begin{aligned} t_P(d,p) &= \begin{cases} \text{number of steps taken} & \\ \text{by } P \text{ to compute } \phi_P(d,p), & \text{if } \phi_P(d,p) \text{ is defined} \\ \text{undefined,} & \text{otherwise} \end{cases} \\ &= \mu t(P(d,p) \text{ stops in } t \text{ steps}). \end{aligned} \tag{1}$$

If e is an index of P, we write $t_e(d, p)$ for $t_P(d, p)$. It is obvious that $\mathrm{Dom}(t_e) = \mathrm{Dom}(\phi_e)$ for all e, and the predicate $\boldsymbol{M}(e, \langle d, p \rangle, y)$ defined by $\boldsymbol{M}(e, \langle d, p \rangle, y) \equiv$ "$t_e(d, p) \simeq y$" is decidable [6].

We say that a predicate $\boldsymbol{Q}(n)$ holds for almost all n, if $\boldsymbol{Q}(n)$ holds for all n but finitely many numbers n. Equivalently, there is a number n_0 such that $\boldsymbol{Q}(n)$ holds for all $n \geq n_0$. This property is simply denoted by $\boldsymbol{Q}(n)$ a.e.(almost everywhere).

III. Definitions of Computer Viruses

In this section, we extend Adleman's definitions about computer viruses [3] to comply with common understanding of computer viruses [7].

Defintion 3.1 (Nonresident Virus): A total recursive function v is called a nonresident virus if for all x, v satisfies the following

a)

$$\phi_{v(x)}(d,p) = \begin{cases} D(d,p), & \text{if } T(d,p) \quad \text{(i)} \\ \phi_x(d,p[v(\underline{S(p)})]), & \text{if } I(d,p) \quad \text{(ii)} \\ \phi_x(d,p), & \text{otherwise (iii).} \end{cases} \tag{2}$$

b) $D(d,p)$ and $S(p)$ are two recursive functions. $T(d,p)$ and $I(d,p)$ are two recursive predicates such that there is at least one pair of (d,p) in which $I(d,p)$ holds, and that there is no (d,p) in which both $T(d,p)$ and $I(d,p)$ hold simultaneously.

c) The set $\{(d,p) : \neg(T(d,p) \vee I(d,p))\}$ is infinite.

$T(d,p)$ and $I(d,p)$ are called *injury* condition (trigger) and *infection* condition, respectively. When $T(d,p)$ holds, the virus executes the *injury* function $D(d,p)$, and when $I(d,p)$ holds, the virus chooses a program by *selection* function $S(p)$, infects it first, then executes the original program x. Conditions $T(d,p)$ and $I(d,p)$ together with functions $D(d,p)$ and $S(p)$ are called the *kernel* of a nonresident virus in the rest of this correspondence, because they determine a nonresident virus uniquely.

In definition 3.1, formula (i), (ii), and (iii) describe three typical types of behavior of computer viruses, namely, *injury*, *infection*, and *imitation*. Condition b) guarantees that an infected program would infect at least one other program, and the infection and injury action cannot be completed simultaneously. In other words, infectivity is an indispensable attribute of computer viruses. Condition c) requires that an infected program *imitates* the original program at infinitely many points. It is a quite strong condition and necessary for the important property that the set of one type of computer viruses with same kernel is $\prod_2$-complete [7]. However, it is not needed in investigating time complexity about computer viruses.

We may define almost all kinds of computer viruses in this way. For example, we can give similar definitions for other kinds of computer viruses which are both important and interesting, theoretically and practically. In these definitions, we no longer list conditions b) and c) as in Definition 3.1, but assume they are included in all of these definitions.

Defintion 3.2 (Polymorphic Virus With Two Forms): The pair (v, v') of two different total recursive functions v and v' is called a polymorphic virus with two forms if for all x, (v, v') satisfies

$$\phi_{v(x)}(d,p) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d,p[v'(\underline{S(p)})]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise} \end{cases} \tag{3}$$

and

$$\phi_{v'(x)}(d,p) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d,p[v(\underline{S(p)})]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise.} \end{cases} \tag{4}$$

Polymorphic viruses with n forms can be defined as an n-tuple $(v_1, v_2, \ldots, v_n)$ of n different total recursive functions, under similar conditions as in Definition 3.2. Polymorphic viruses have spread widely in the last ten years and caused a lot of trouble as they are very hard to detect. Polymorphic viruses have billions of forms and in general any two forms do not have three consecutive bytes in common. However, polymorphic viruses are not the hardest viruses to detect. Two other kinds of viruses, the polymorphic viruses with infinite forms and metamorphic viruses [8] are a real challenge.

Defintion 3.3 (Polymorphic Virus With Infinite Forms): A total recursive function $v(m,x)$ is called a polymorphic virus with infinite forms if for all m and x, $v(m,x)$ satisfies

$$\phi_{v(m,x)}(d,p) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d,p[v(m+1,\underline{S(p)})]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise} \end{cases} \tag{5}$$

and for all $m \neq n$, $v(m,x) \neq v(n,x)$.

Up to now polymorphic viruses with infinite forms have not been found in the real world, but their existence can be proved by the recursion theorem of [7] just like ordinary computer viruses.

Defintion 3.4 (Metamorphic Virus): The pair (v, v') of two different total recursive functions v and v' is called a metamorphic virus if for all x, (v, v') satisfies

$$\phi_{v(x)}(d,p) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d,p[v'(\underline{S(p)})]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise} \end{cases} \tag{6}$$

and

$$\phi_{v'(x)}(d,p) = \begin{cases} D'(d,p), & \text{if } T'(d,p) \\ \phi_x(d,p[v(\underline{S'(p)})]), & \text{if } I'(d,p) \\ \phi_x(d,p), & \text{otherwise} \end{cases} \tag{7}$$

where $T(d,p)$ (resp., $I(d,p)$, $D(d,p)$, $S(p)$) is different from $T'(d,p)$ (resp., $I'(d,p)$, $D'(d,p)$, $S'(p)$).

A metamorphic virus (v, v') seems to combine two different computer viruses v and v'. But, there is a crucial distinction between a metamorphic virus and a pair of two viruses, that is, when v infects a program, the program is indeed infected by v' (not v), and *vice versa*. The metamorphic virus $(v_1, v_2, \ldots, v_n)$ can be defined in the similar way. The main difference between a metamorphic virus and a polymorphic virus is that each form of a polymorphic virus has the same kernel, but each component v_n of a metamorphic virus has its own kernel [8].

More discussions about computer viruses can be found in [7].

IV. Main Results

In this section, we prove the main results of this correspondence, using the traditional notations and symbols in recursive function theory([9], [10]).

Lemma 4.1: Let R be an infinite recursive set and $b(x)$ be a total recursive function. Then there exists a recursive subset $A \subseteq R$ such that $t_e(x) > b(x)$ a.e., where e is any index of the characteristic function of A.

Proof: Let g be an increasing total recursive function with $E_g = R$. Define

$$i_{g(y)} = \begin{cases} \mu i[i \le y \text{ and } i \text{differs from} \\ \text{all previously defined } i_k & \text{if such an } i \text{ exists} \\ \text{and } t_i(y) \le b(y)], \\ \text{undefined,} & \text{otherwise.} \end{cases} \tag{8}$$

Since

$$t_i(y) \le b(y) \Leftrightarrow \exists z \le b(y)(t_i(y) \simeq z) \tag{9}$$

and "$t_i(y) \simeq z$" is a recursive predicate, there is an effective procedure to decide whether $i_{g(y)}$ is defined, and to compute its value when it is defined. Consider the function

$$f(x) = \begin{cases} 1, & \text{if } x = g(y) \text{ for some } y, \\ & i_{g(y)} \text{ is defined and } \phi_{i_{g(y)}}(y) = 0 \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Since R is an recursive set, $f(x)$ is a well-defined total recursive function. Let $\phi_e = f$, by diagonal construction of $f(x)$, $e \neq i_{g(x)}$ whenever $i_{g(x)}$ is defined.

Now, for any c such that $t_c(x) \leq b(x)$ for infinitely many x, let

$$s = 1 + \max\{m : i_{g(m)} \text{ is defined and } i_{g(m)} < c\}$$

($s = 0$ if m does not exist). Choose p such that $p \geq \max\{c, s\}$ and $t_c(p) \leq b(p)$. If $c = i_{g(m)}$ for some $m < p$, there is nothing to prove. Assuming $c \neq i_{g(m)}$ for all $m < p$, we have

$$c \leq p \text{ and } c \text{ differs from all previously defined } i_m \text{ and } t_c(p) \leq b(p). \tag{11}$$

Hence, c is the least index satisfying (11). From the definition (8) we know that $i_{g(p)}$ is defined and equals c. Thus, we have obtained that for any c satisfying $t_c(x) \leq b(x)$, for infinitely many x, there exists a y such that $c = i_{g(y)}$. Since $e \neq i_{g(y)}$ whenever $i_{g(y)}$ is defined, it follows that $t_e(x) > b(x)$ a.e.

Let $A = \{x : f(x) = 1\}$. A is a recursive set, and $A \subseteq R$. □

Roughly speaking, Lemma 4.1 implies that for any infinite recursive set, there exists a recursive subset whose decision procedures have arbitrarily large time complexity.

Corollary 4.1: For any total recursive function $b(x)$, there exists a recursive set A such that $t_e(x) > b(x)$ a.e., where e is any index of characteristic function defined by

$$f(x) = \begin{cases} 1, & x \in A \\ 0, & x \notin A. \end{cases} \tag{12}$$

Proof: Let $R = \mathbb{N}$, by Lemma 4.1 we have the conclusion. □

Corollary 4.2: Let $b(x)$ be a total recursive function and R be recursively enumerable but not a simple set. Then there exists a recursive set $C \supseteq R$ such that $t_e(x) > b(x)$ a.e., where e is any index of characteristic function of C.

Proof: Since R is recursively enumerable and not simple, its complement $\overline{R}$ contains an infinite recursively enumerable set B that again has an infinite recursive subset B'. Applying Lemma 4.1 to B', there is a recursive subset A, $A \subseteq B' \subseteq B \subseteq \overline{R}$ such that $t_e(x) > b(x)$ a.e., where e is any index of characteristic function of A. Let $C = \overline{A}$, then C is the set required. □

Lemma 4.2: Let $b(x)$ be a total recursive function. For any recursive function $f(x, y, z)$, there exists a total recursive function $k(x, y)$ such that $\phi_{k(x,y)}(z) = f(x, y, z)$ and $t_e(x, y) > b(x)$ a.e. for all y, where e is any index of $k(x, y)$.

Proof: By the s–m–n theorem, there is a total recursive function r such that

$$c\phi_{r(x,y,s)}(z) = \mathbf{1}(s) f(x, y, z) \tag{13}$$

and $r(x, y, s)$ satisfies $r(x, y, s_1) \neq r(x, y, s_2)$ (if $s_1 \neq s_2$) [6].

Define

$$k(x, y) = \begin{cases} r(x, y, 1), & \text{if } i_x \text{ is defined} \\ & \text{and } \phi_{i_x}(x) = r(x, y, 0) \\ r(x, y, 0), & \text{otherwise} \end{cases} \tag{14}$$

where i_x is given by (8) for $R = \mathbb{N}$ and $g(y) = y$. From the proof of Lemma 4.1, $k(x, y)$ is a total recursive function and $t_e(x, y) > b(x)$ a.e. for all y, where e is any index of $k(x, y)$.

Since

$$\begin{aligned} \phi_{k(x,y)}(z) &= \begin{cases} \phi_{r(x,y,1)}(z), & \text{if } i_x \text{ is defined} \\ & \text{and } \phi_{i_x}(x) = r(x, y, 0) \\ \phi_{r(x,y,0)}(z), & \text{otherwise} \end{cases} \\ &= \begin{cases} \mathbf{1}(1) f(x, y, z), & \text{if } i_x \text{ is defined} \\ & \text{and } \phi_{i_x}(x) = r(x, y, 0) \\ \mathbf{1}(0) f(x, y, z), & \text{otherwise} \end{cases} \\ &= f(x, y, z) \end{aligned} \tag{15}$$

this completes the proof of the lemma. □

Lemma 4.2 extends the s–m–n theorem in the sense that the index function $k(x, y)$ could be any recursive function, not just the primitive recursive function as in the s–m–n theorem. Lemma 4.1 can be further extended as follows.

Lemma 4.3: For each $m_1, m_2, n \geq 1$, let $m = m_1 + m_2$ and $b(\boldsymbol{x})$ be a total recursive m_1-ary function. There exists a total recursive $(m + 1)$-ary function $s_n^m(c, \boldsymbol{x}, \boldsymbol{y})$ such that

$$\phi_c^{(m+n)}(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = \phi_{s_n^m(c,\boldsymbol{x},\boldsymbol{y})}^{(n)}(\boldsymbol{z})$$

and $t_e(\boldsymbol{x}, \boldsymbol{y}) > b(\boldsymbol{x})$ a.e. for all $\boldsymbol{y}$, where e is any index of $s_n^m(c, \boldsymbol{x}, \boldsymbol{y})$.

Lemma 4.3 implies that for any type of computer viruses, there exists a computer virus v whose infecting procedure has arbitrarily large time complexity. This conclusion is formulated in the following theorem.

Theorem 4.1: Let $b(x)$ be a total recursive function. For any kind of computer viruses, there exists a computer virus $v(x)$ such that $t_e(x) > b(x)$ a.e., where e is any index of $v(x)$.

Proof: Let $b(x)$ be a total recursive function. Consider

$$f(x, k, \langle d, p\rangle) = \begin{cases} D(d, p), & \text{if } T(d, p) \\ \phi_x(d, p[\phi_k(\underline{S(p)})]), & \text{if } I(d, p) \\ \phi_x(d, p), & \text{otherwise.} \end{cases} \tag{16}$$

By Lemma 4.3, there is a total recursive function $h(x, k)$ such that

$$\phi_{h(x,k)}(d, p) = f(x, k, \langle d, p\rangle) \tag{17}$$

and $t_e(x, k) > b(x)$ a.e. for all k, where e is any index of $h(x, k)$.

By the s–m–n theorem, there exists a total recursive function $r(k)$ such that $\phi_{r(k)}(x) = h(x, k)$. By recursion theorem, there is an n such that $\phi_{r(n)} = \phi_n$. Let $v(x) = h(x, n) = \phi_{r(n)}(x) = \phi_n(x)$, then

$$\begin{aligned} \phi_{v(x)}(d, p) &= \phi_{h(x,n)}(d, p) = f(x, n, \langle d, p\rangle) \\ &= \begin{cases} D(d, p), & \text{if } T(d, p) \\ \phi_x(d, p[\phi_n(\underline{S(p)})]), & \text{if } I(d, p) \\ \phi_x(d, p), & \text{otherwise} \end{cases} \\ &= \begin{cases} D(d, p), & \text{if } T(d, p) \\ \phi_x(d, p[v(\underline{S(p)})]), & \text{if } I(d, p) \\ \phi_x(d, p), & \text{otherwise.} \end{cases} \end{aligned} \tag{18}$$

By Definition 3.1, the total recursive function $v(x)$ is a nonresident virus and $t_e(x) > b(x)$ a.e., where e is any index of $v(x)$. Similar results also hold for other kinds of computer viruses defined in Section III and [7]. This completes the proof of the theorem. □

Intuitively, we can construct a virus with arbitrarily large time complexity in its infection procedure by adding time-consuming operations in the procedure. But Theorem 4.1 implies more than that. Since by our definition a virus $v(x)$ is a recursive function mapping a program x into its infected form $v(x)$, Theorem 4.1 shows that for any kind of computer viruses there is a virus v such that any implementation of v can have arbitrarily large time complexity in its infection procedure.

It is natural to ask whether there exists a computer virus v such that the infected program $v(x)$ has arbitrarily large time complexity for any program x. In general the answer is NO, because ϕ_x may be a partial recursive function, and if so, by infinite imitation requirement (see c) in Definition 3.1) the function $\phi_{v(x)}(d,p)$ computed by the infected program $v(x)$ is also a partial recursive function, as well as the function $t_{v(x)}(d,p)$. Hence, the comparison with a total recursive function $b(d,p)$ may be meaningless at infinitely many points. But for total recursive functions we have the following result.

Theorem 4.2: Let $b(x)$ be a total recursive function. There exists a computer virus $v(x)$ such that $t_{v(x)}(d,p) > b(d,p)$ a.e. for any total recursive function ϕ_x.

Proof: For each $x, k \in \mathbb{N}$, consider

$$f(x,k,\langle d,p\rangle) = \begin{cases} 1, & \text{if } i_{\langle d,p\rangle} \text{ is defined and } \phi_{i_{\langle d,p\rangle}}(d,p)=0 \\ 0, & \text{if } i_{\langle d,p\rangle} \text{ is defined and } \phi_{i_{\langle d,p\rangle}}(d,p)\neq 0 \\ \phi_x(d,p[\phi_k(\underline{(p)_1})]), & \text{if } i_{\langle d,p\rangle} \text{ is undefined and } (d)_1=0 \\ \phi_x(d,p), & \text{otherwise} \end{cases} \tag{19}$$

where $i_{\langle d,p\rangle}$ is defined in (8) with $R=\mathbb{N}$ and $g(d,p)=\langle d,p\rangle$. By the proof of Lemma 4.1, $f(x,k,\langle d,p\rangle)$ is a recursive function, and if ϕ_x is a total recursive function and e is an index of f, then $t_e(d,p) > b(d,p)$ for almost all $\langle d,p\rangle$.

From the proof of Theorem 4.1, there is a total recursive function $v(x)$ satisfying

$$\phi_{v(x)}(d,p) = \begin{cases} 1, & \text{if } i_{\langle d,p\rangle} \text{ is defined and } \phi_{i_{\langle d,p\rangle}}(d,p)=0 \quad \text{(i)} \\ 0, & \text{if } i_{\langle d,p\rangle} \text{ is defined and } \phi_{i_{\langle d,p\rangle}}(d,p)\neq 0 \quad \text{(i')} \\ \phi_x(d,p[v(\underline{(p)_1})]), & \text{if } i_{\langle d,p\rangle} \text{ is undefined and } (d)_1=0 \quad \text{(ii)} \\ \phi_x(d,p), & \text{otherwise} \quad \text{(iii).} \end{cases} \tag{20}$$

By Definition 3.1, the total recursive function $v(x)$ is a nonresident virus. (In (20), condition (i) and (i′), (ii), (iii) denote injury, infection, and imitation of nonresident viruses, respectively.)

Since $v(x)$ is an index of the recursive function f, it follows that $t_{v(x)}(d,p) > b(d,p)$ a.e. for total recursive function ϕ_x. □

Virus detection is one of the most important issues in antivirus practice. It is well known that the set of all computer viruses is undecidable [1]. Furthermore, there exists a computer virus v such that the set of its infected programs is undecidable [3], [11]. In other words, we can never find a procedure to pick up exactly all the programs infected by v. Formally, let $I_v = E_v = \{v(x) : x \in \mathbb{N}\}$ [3], then I_v is a nonrecursive recursively enumerable set. To find all programs infected by v, it is necessary to find a recursive set C such that $I_v \subset C$. This implies that for every detecting procedure of v, if it has no false negatives, then it always has false positives.

Although most of the computer viruses in real world are decidable, there are still two unresolved questions. 1) If I_v is decidable, what is its time complexity? 2) If I_v is undecidable, what is the time complexity of the recursive set containing I_v? The following theorem gives a partial answer to the first question.

Theorem 4.3: Let $b(x)$ be a total recursive function, then there exists a decidable computer virus v such that $t_e(m) > b(m)$ for infinitely many m, where e is any index of characteristic function of I_v.

Proof: Let A be an infinite recursive set and g be an increasing recursive function with $E_g = A$. Consider

$$h(x,k,y,\langle d,p\rangle) = \begin{cases} f(x,k,m,\langle d,p\rangle), & \exists m.y=g(m) \\ \boldsymbol{Id}, & \text{otherwise} \end{cases} \tag{21}$$

where $\boldsymbol{Id}$ is the identity function, and f is defined by

$$f(x,k,m,\langle d,p\rangle) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d,p[\phi_k(g(m+1),\underline{S(p)})]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise.} \end{cases} \tag{22}$$

From the proof of Theorem 4.1, there is a recursive function $s(x,y)$ such that

$$\phi_{s(x,y)}(d,p) = \begin{cases} f'(x,m,\langle d,p\rangle), & \exists m.y=g(m) \\ \boldsymbol{Id}, & \text{otherwise} \end{cases} \tag{23}$$

and

$$f'(x,m,\langle d,p\rangle) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d,p[s(g(m+1),\underline{S(p)})]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise.} \end{cases} \tag{24}$$

Moreover, $s(x,y)$ is an increasing function [10]. Let $v(m,x) = s(g(m),x)$, then

$$\phi_{v(m,x)}(d,p) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d,p[v(m+1,\underline{S(p)})]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise.} \end{cases} \tag{25}$$

By Definition 3.3, v is a polymorphic virus with infinite forms.

Let $I_v = \{v(m,x) | m,x \in \mathbb{N}\}$. Since g and s are increasing functions, v is also an increasing function. This implies that v is a decidable virus and I_v is recursive. Let $r(m) = v(m,n)$, then $m \in A \Leftrightarrow r(m) \in I_v$. By Corollary 4.1, there is a set A such that $t_{e'}(x) > b(r(x))$ a.e., where e' is any index of the characteristic function of A.

Now suppose c is the characteristic function of I_v and e is one of its indices, since $c(r(x))$ is the characteristic function of A, $t_e(r(x) > b(r(x))$ a.e.. Let $m = r(x)$, this completes the proof of the theorem. □

When I_v is undecidable, we should consider the time complexity of the recursive sets containing I_v. The following theorem shows that for any undecidable computer virus, there is one detecting procedure which has arbitrarily large time complexity.

Theorem 4.4: Suppose $v(x)$ is a computer virus and I_v is a nonrecursive recursively enumerable set. For any total recursive function $b(x)$ there exists a recursive set $C \supset I_v$ such that $t_e(x) > b(x)$ a.e., where e is any index of the characteristic function of C.

Proof: Let v be a computer virus and $\boldsymbol{Id}$ be the identity function. By Rice's theorem, $\{i | \phi_i \equiv \boldsymbol{Id}\}$ is a recursively enumerable set. By Definition 3.1 b), $\phi_{v(x)}$ is not an identify function for all x. This implies if i is any index of $\boldsymbol{Id}$, then $i \notin I_v$. Hence, $\{i|\phi_i \equiv \boldsymbol{Id}\} \cap I_v = \emptyset$ and I_v is not a simple set. By Corollary 4.2 we have the conclusion. □

If v is an undecidable computer virus, that is, I_v is a nonrecursive recursive enumerable set, then $(C - I_v)$ is infinite for any recursive set C containing I_v. This implies that any of its detecting procedures which can pick up all of its infected program, will make infinitely many errors (false positives). However, it is often desired to find the detecting procedure that gives "minimal" errors. In other words, we want to find a minimal recursive set C such that $I_v \subset C$. Here a recursive set C containing A is called minimal means that for any recursive set B such that $A \subset B \subset C$, the set $(C - B)$ is finite. In the following theorem, we prove that this desired minimal recursive set does not exist for some computer viruses. The proof, inspired by Adleman [3], depends on the following lemma.

Lemma 4.4: Let A be a creative set. Then there is no minimal recursive set containing A.

Proof: Let $p(x)$ be the productive function of $\overline{A}$. If x satisfies $W_x \cap A = \emptyset$, then clearly $p(x) \notin W_x$ and $p(x) \notin A$. By the s–m–n theorem, let g be a recursive function such that for all x

$$W_{g(x)} = W_x \cup \{p(x)\}.$$

Suppose C is a recursive set such that $A \subset C$, and let $D = \overline{C}$. Since D is recursively enumerable, $D = W_e$ for some e. Now let

$$H = \{p(e), g(p(e)), g(g(p(e))), \ldots\}.$$

Clearly H is recursively enumerable. Since $p(e), g(p(e)), g(g(p(e))), \ldots$ are pairwise distinct, H is infinite and has some infinite recursive subset J. Also, by the above properties, $H \cap D = H \cap A = \emptyset$. Hence, $J \subseteq C$ and $A \subseteq \overline{J}$. Let $B = C - J$, then B is recursive and $A \subset B \subset C$, but $(C - B)$ is infinite. □

Theorem 4.5: There exists computer virus v such that I_v is nonrecursive and there is no minimal recursive set containing it.

Proof: Let $j(i, y)$ be an increasing padding function, i.e., for all i and y, $\phi_{j(i,y)}(x) = \phi_i(x)$. Let $\boldsymbol{K} = \{e : e \in W_e\}$ and g be the total recursive function such that $\boldsymbol{K} = E_g$. Consider the function

$$h(i) = \begin{cases} j(1, g(y)), & i = j(1, y) \\ j(i, g(0)), & \text{otherwise.} \end{cases} \tag{26}$$

It is clear that $\phi_{h(i)}(x) = \phi_i(x)$. Let $c(y) = j(1, y)$, since j is a one-to–one function, it follows that

$$y \in \boldsymbol{K} \Leftrightarrow c(y) \in E_h. \tag{27}$$

Now let $f(x, k)$ be the one-to-one total recursive function such that

$$\phi_{f(x,k)}(d,p) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d, p[\phi_k(h(\underline{S(p)}))]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise.} \end{cases} \tag{28}$$

From the proof of Theorem 4.1, there exits a total function $s(x)$ such that

$$\phi_{s(x)}(d,p) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d, p[s(h(\underline{S(p)}))]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise.} \end{cases} \tag{29}$$

Substituting x by $h(x)$ in (29), it follows that

$$\phi_{s(h(x))}(d,p) = \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_{h(x)}(d, p[s(h(\underline{S(p)}))]), & \text{if } I(d,p) \\ \phi_{h(x)}(d,p), & \text{otherwise} \end{cases}$$

$$= \begin{cases} D(d,p), & \text{if } T(d,p) \\ \phi_x(d, p[s(h(\underline{S(p)}))]), & \text{if } I(d,p) \\ \phi_x(d,p), & \text{otherwise.} \end{cases} \tag{30}$$

Let $v = sh$, then v is a nonresident virus by Definition 3.1. Since s is a one-to-one total function, $x \in E_h \Leftrightarrow s(x) \in I_v$. Combined with (27) we have

$$y \in \boldsymbol{K} \Leftrightarrow s(c(y)) \in I_v \tag{31}$$

i.e., $\boldsymbol{K} \leq_1 I_v$. It means I_v is 1-complete set, hence equivalently creative set. From Lemma 4.4, it follows that there is no minimal recursive set containing I_v. □

V. Conclusion

In this correspondence, we discussed the time complexity of computer viruses. The main contributions are as follows.

1) We proved that there are computer viruses with arbitrarily large time complexity, not only in their infecting procedures, but also in their executing procedures.
2) We proved that there are computer viruses whose detecting procedures have sufficiently large time complexity.
3) We proved that there are undecidable viruses which have no minimal detecting procedure.

It is worth noting that in our discussions we used only the following two features of t_e:

1) $\mathrm{Dom}(t_e) = \mathrm{Dom}(\phi_e)$ for all e; and
2) "$t_e(x) \simeq y$" is decidable.

If we take these two conditions as axioms (as in [5]), all of the conclusions on time complexity of computer viruses can be extended directly to other computational complexity satisfying these two assumptions, such as space complexity of computer viruses, etc.

Contrary to the conclusion of Theorem 4.4, in antivirus practice we are concerned more about the existence of a recursive set C satisfying $I_v \subseteq C$ and the characteristic function of C have "low" time complexity, such as linear or polynomial time complexity. This problem is trivial when $C = \mathbb{N}$, but if we require that $(\mathbb{N} - C)$ is infinite and C is as small as possible, it is an open problem.

For example, typical pattern-based virus-detection software is usually based on the Boyer–Moore string-searching algorithm [12] (or similar algorithms), and can detect most of simple computer viruses with false positives in linear time. But it is not clear whether there are linear or polynomial algorithms which can work for all kinds of computer viruses, especially for undecidable computer viruses.

Acknowledgment

The authors wish to thank the referees for their helpful comments that improved the correspondence greatly.

References

[1] F. Cohen, "Computational aspects of computer viruses," *Computers & Security*, vol. 8, no. 1, pp. 325–344, 1989.
[2] ——, *A Short Course on Computer Viruses*. New York: Wiley, 1994.
[3] L. M. Adleman, "An abtract theory of computer viruses," in *Advances in Cryptology–CRYPTO'88 (Lecture Notes in Computer Science)*, S. Goldwasser, Ed. Berlin, Germany, 1988, vol. 403, pp. 354–374.
[4] D. Spinellis, "Reliable identification of bounded-length viruses is NP-complete," *IEEE Trans. Inf. Theory*, vol. 49, no. 1, pp. 280–284, Jan. 2003.
[5] M. Blum, "A machine-independent theory of the complexity of recursive function," *J. ACM*, vol. 14, no. 2, pp. 322–336, 1967.
[6] N. Cutland, *Computability: Introduction to Recursive Function Theory*. Cambridge, U.K.: Cambridge Univ. Press, 1980.
[7] Z. H. Zuo and M. H. Zhou, "Some further theoretical results about computer viruses," *Comp. J.*, vol. 47, no. 6, pp. 625–633, 2004.
[8] P. Ször and P. Ferrie. (2000) Hunting for Metamorphic. [Online]. Available: http://www.virusbtn.com
[9] H. J. Rogers, *Theory of Recursive Functions and Effective Computability*. New York: McGraw-Hill, 1967.
[10] R. I. Soare, *Recursively Enumerable Sets and Degrees*. New York: Springer-Verlag, 1987.
[11] D. M. Chess and S. R. White, "An undetectable computer virus," in *Proc. Virus Bulletin Conf.*, Orlando, FL, Sep. 2000.
[12] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 262–272, Oct. 1977.