Journal of Siberian Federal University. Mathematics & Physics 2011, 4(2), 195–207

УДК 510.52

Analysis Parameterized Algorithms on the Bases of Elasticity to Functions Complexity

Valentina V. Bykova*
Institute of Mathematics,
Siberian Federal University,
Svobodny, 79, Krasnoyarsk, 660041
Russia

Received 30.10.2010, received in revised form 10.11.2010, accepted 20.12.2010

We give a brief overview of results and problems of parameterized algorithmics as the new direction of computational complexity theory. We offer a new indicator of computational complexity for parameterized algorithm which can be used to measure rate a growth of function complexity from many variables. This indicator is a private elasticity of the function complexity. We offer a two-dimensional classification parameterized algorithms to multiplicative forms a presentation of the functions complexity. We give a mathematical basis to analysis a level impact of parameter for time execution of parameterized algorithm.

Keywords: computation complexity, parameterized algorithms, analysis algorithms, elasticity algorithms.

Introduction

According to the classical complexity theory, many problems that have important real world applications are NP-hard [1]. To deal with NP-hard problems, many approaches have been proposed. Approximation algorithms and parameterized algorithms are two of these approaches. Objective of approximation algorithms is to find in polynomial time solution which is close to the optimal solution. A notable and important class of approximate algorithms for solving NP-hard combinatorial optimization problems is fully polynomial-time approximation schemes [2]. Parameterized algorithms try to give exact solutions for NP-hard problems when its natural parameter k is small even if dimension n of the problem (length input data for algorithm) is big. In parameterized complexity theory measure complexity takes into account not only the n, but the numerical parameter k, whose value may depend on n arbitrary way or at all not depend on n [3]. Roles of parameter take into account the information about structure of input data and identify the main sources nonpolynomial complexity of NP-hard problem. Parameterized complexity theory uses two-dimensional functions of computational complexity of algorithms, so it is also called a two-dimensional complexity theory. Whereas the classical (one-dimensional) complexity theory analyzes and classifies problems and algorithms only in terms of resource (time or memory), which is required for the execution of algorithm depending on size of the input data [1]. Classical one-dimensionality and orientation in analysis of algorithms on the worst case often makes the problems more complicated than they really are. Parameterized complexity theory provides a basis for detailed analysis of the complexity problems that are intractable in classical sense. Parameterized complexity is a fairly new branch of complexity theory. It was developed by Downey and Fellows in the early 1990s [4]. Over past two decades the ideas from parameterized complexity theory have found their way into various areas of computer science, such as artificial intelligence [5], computational biology [6], database theory [7, 8] and etc.

*bykvalen@mail.ru

Parameterized Problems and Algorithms

The parameterized complexity theory investigates only of the decision problem, i.e. problem, in which we must answer "yes" or "no" to some question. Generality is not lost. This approach is also used in theory of NP-completeness [1].

We give below basic concepts of parameterized complexity theory [3, 4].

A *parameterized problem* Π is language which is defined as $L(\Pi) \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet, Σ^* is set of all words in Σ , $\mathbb{N}$ is set of all nonnegative integers. If $(I, k) \in L(\Pi)$, then I is called the *main part* and integer number k is called the *parameter* of parameterized problem Π . Each particular pair of values (I, k) is an instance of problem and is also input data for algorithm α that solves this problem. The value $|I| = n$ specifies a length the input data for algorithm α (*dimension* of parameterized problem Π).

We say that an *algorithm α solves the parameterized problem* Π, if for each input (I, k) the algorithm α can determine whether (I, k) is a yes-instance of $L(\Pi)$ (i.e. whether (I, k) is an element of $L(\Pi)$).The algorithm α is said to be *parameterized algorithm*, if its computational complexity as the need for resources (further only execution time of algorithm) is measured in terms of both input length $|I| = n$ and value of parameter k. Thus, the function complexity of parameterized algorithm that determines the execution time of algorithm is a function of two variables $t(n, k)$.

A parameterized problem Π is *fixed-parameter tractable problem* (FPT-problem), if it can be solved by some parameterized algorithm α in time

$$t(n, k) = O(n^{O(1)} \cdot f(k)) \tag{1}$$

for some function f depending only on k. The growth rate of function $f(k)$ is unlimited. For example, $f(k) = 2^{o(k)}$ or $f(k) = 2^{O(k)}$. It is important that we eliminate the function of form $f(n, k)$, for example, $f(n, k) = n^k$. Denote by FPT the class of all fixed-parameter tractable problems. Appropriate parameterized algorithms, that solve such problems, are called FPT-*algorithms*.

An example, FPT-problem is a problem SATISFIABILITY of formulas propositional logic when parameter k is by the number of variables. In fact, a formula of size n with k variables can be tested by algorithm of brute force in time $O(n \cdot 2^k)$. A vertex cover of size k in a graph G with n vertices can also be found by brute force in time $O(n \cdot 2^k)$. Therefore, problem VERTEX COVER is FPT-problem where the parameter is the size of the cover. If in problem VERTEX COVER as a parameter to take the treewidth $tw(G)$ of graph G then dynamic programming technique finds a vertex cover of size at most $tw(G)$ in time $O(n \cdot 2^{tw(G)})$ [9]. Consequently, a parameterization of problem VERTEX COVER with respect $tw(G)$ leads to FPT.

An example, a problem, which does not belong to FPT, is problem GRAPH COLORING (k-coloring of n-vertex graph) with parameter k. It is known that a problem of 2-coloring can be solved in time $O(n^{O(1)})$. However, 3-coloring is NP-hard problem with respect n [1].

Obviously, FPT contains all polynomial-time computable problems. However, the greatest interest in terms of theory and practice are of FPT-solvable problems that are NP-hard. From a theoretical point of view all these problems can be solved in polynomial time for each fixed parameter value. Meanwhile, really it is possible to implement in most cases only for small values of parameter (it all depends on type of function $f(k)$). Already there are collections of parameterized problems, including fixed-parameter tractable problems. The most concrete at the moment is a collection Marco Chezati [10].

The parameterized complexity theory is developing on several directions: definition of a class hierarchy of parameterized problems, identification conditions of belonging to the class FPT, identification the relationship between of parameterized complexity and classes of approximate algorithms (in particular, fully polynomial-time approximation schemes), development of parameterized algorithmics (development a methods analysis and design of parameterized algorithms)

and etc. [3]. A parameterized algorithmics is the applied direction of parameterized complexity theory. Here there are a number of open questions.

Some Problems of Parameterized Algorithmics

Analysis of the collection parameterized problems [10] shows that a problem might have different parameterizations with respect to various parameters. An example, in problem VERTEX COVER as a parameter can be size of cover, the number of vertices, the number of edges, treewidth, the maximum degree of vertices to graph, etc. The following questions are natural.

1) What can serve as a parameter for NP-hard problem?

2) How to choose parameter so that a parameterized problem belonged to FPT?

3) Are there special methods for developing FPT-algorithms?

4) How to compare various FPT-algorithms of solving some NP-hard problem?

At some of these questions in parameterized complexity theory not given yet a comprehensive responses. For example, a parameterized complexity theory responds on first two questions follows.

There are internal and external parameters. The parameter is *explicit internal parameter*, if a parameter appears directly in main part of instance (I, k) for parameterized problem Π and its values are bounded by a polynomial respect $|I| = n$. Such parameters for problem VERTEX COVER are size of cover, number of vertices or number of edges to graph. There are *standard parameters*. They are most popular. A standard parameter for minimization problems traditionally is a quantity which is necessary to minimize. For problem VERTEX COVER it is size of cover. If $k = n$ then k becomes *trivial parameter* and parameterized algorithm degenerates into usual algorithm.

Parameter is an *implicit internal parameter*, if its values are bounded by a polynomial respect $|I| = n$ and information about this parameter is implicitly hidden in I. A typical example: treewidth of graph (values treewidth always no greater than a number of vertices to graph) is an implicit internal parameter. Altogether, all internal parameters characterize in some way a structure of input data for algorithm.

Finally, there are *external parameters*, which not depend on $|I| = n$ or depend nonpolynomial. Information on these parameters we specify in addition to I, because considered that it no is in I.

From a theoretical point as a parameter can serve any part input data of algorithm. In practice to get good parameterization (for example, related to FPT) is recommended to select such a part input data of algorithm, which usually takes small values as compared with $|I| = n$. Ultimately, it all depends on application. For different applications may be acceptable different parameterization of NP-hard problem. Thus, the second question as a whole remains until theoretically open. For some NP-hard problems is proved that individual parameterization of these problems not can be implemented FPT-algorithms. As mentioned earlier, this is a problem GRAPH COLORING (k-coloring of n-vertex graph) with parameter k. The some NP-hard problems have the FPT-algorithms to individual parameters. Example, for many optimization problems on graphs, treewidth proved to be such parameter which usually leads to FPT-algorithms [9–11].

As regards the third of above questions, then modern theoretical and practical results provide tools are sufficient for algorithmic practice. Currently a toolbox for construction of parameterized algorithms contains set special methods to creating FPT-algorithms. Above all, this dynamic

programming technique based on tree decomposition for graphs with bounded treewidth [11]. Another known method of construction FPT-algorithms is method of parametric reduction (kernelization) [4]. A method of parametric reduction is transformation in polynomial time of each instance (I, k) of parameterized problem Π into instance (I', k') of parameterized problem Π' so that $k' \leq k$. After this for the instance (I', k'), which is called the kernel for I, is used brute force. Example, for problem VERTEX COVER with standard parameter k (size of cover) a method of parametric reduction is reduced to a repeated application of two rules: deletion isolated vertices without changing parameter k, deletion a vertex whose degree is more to k and reduction a parameter k on one. It should be noted, that a method of parametric reduction combined with brute force does not always lead to FPT-algorithm. Meanwhile it is proved, that if a problem belongs to FPT then this method invariably gives FPT-algorithm [4].

Fourth question is theoretically open: in numerous papers on parameterized complexity theory to analysis and compare of parameterized algorithms apply methods of classical (one-dimensional) complexity theory. This is predominantly different approaches to defining the functions complexity and study asymptotic behavior of these functions for large values n [12–14]. Meanwhile, classical one-dimensionality limits depth of analysis parameterized algorithms, since for them the computational complexity described by function $t(n, k)$ on two variables.

In this paper we propose a measure of computational complexity of the algorithm which can be used for analysis rate a growth of functions to several variables, for analysis a level impact of structure of input data on the computational complexity parameterized algorithms, for comparison of FPT-algorithms solving the NP-hard problems. This measure is the elasticity to function complexity of algorithm. Presented in this paper results are a generalization and development of author's results, published in [15–17] for the one-dimensional complexity theory.

Agreements Concerning Functions Complexity

A formal approach to analysis of parameterized algorithms requires assumption of some properties a functions complexity $t(n, k)$ of algorithms. By respect functions $t(n, k)$ we make some natural assumptions, which are feasible for most real algorithms and which are necessary for application of apparatus mathematical analysis:

- we believe that $t(n, k)$ is a monotonous nondecreasing function respect both arguments. The range of values this function is the set of nonnegative real numbers and the domain of definition a function is the set $\mathbb{N} \times \mathbb{N}$;

- we deviate from discreteness n and k (with formal replacement of n by x, of k by y), i.e. we believe that arguments of function $t(n, k)$ are continuous and necessary values are computed at integer points $x = n$ and $y = k$. According to this assumption, below instead of $t(n, k)$ we write $z(x, y)$;

- we assume that set under consideration of functions is limited family $\mathfrak{L}$ of "basically positive" logarithmic-exponential functions. A family $\mathfrak{L}$ was introduced and studied by G. Hardy [18]. Recall that $z(x, y)$ is "basically positive" function, if there exists x_0, y_0 such, that $z(x, y) > 0$ for all $x > x_0$, $y > y_0$. It is known that each such function is continuous and differentiable in domain where it is defined.

These assumptions are the guarantee existence of elasticity for functions complexity of parameterized algorithms. Henceforth, we mean that the $\mathfrak{L}$-function always is a monotonous nondecreasing, "basically positive", logarithmic-exponential function of family $\mathfrak{L}$.

Elasticity as a Measure Computational Complexity of Algorithm

A elasticity $E_x(z)$ *to function* $z = z(x)$ is the limit ratio of relative increment this function to relative increment of argument [15]:

$$E_x(z) = \lim_{\Delta x \to 0} \left(\frac{\Delta z}{z} : \frac{\Delta x}{x} \right) = \frac{dz}{z} : \frac{dx}{x} = \frac{dz}{dx} \cdot \frac{x}{z} = z' \frac{x}{z} = x(\ln z)'. \tag{2}$$

It is obvious, that for any $\mathfrak{L}$-function $z = z(x)$ always exist a elasticity and $E_x(z) \geq 0$. In [16] we showed that when $x \to \infty$ the elasticity of $\mathfrak{L}$-functions is identically constant, infinitely small or infinitely large logarithmic-exponential function for which the estimates are correct:

$$E_x(z) = o\left(\frac{z}{\ln x}\right), \quad E_x(cz^m) = O[E_x(z)], \quad c > 0, \quad m > 0.$$

Last estimate means that when $x \to \infty$ behavior of elasticity is invariant with respect to a polynomial transformation, which consists of multiplying function by constant $c > 0$ and exponentiation $m > 0$.

It is known that two arbitrary $\mathfrak{L}$–functions $z_1(x)$ and $z_2(x)$ is always comparable in order of growth [18]: if $z_1(x), z_2(x) \in \mathfrak{L}$ then for $x \to \infty$ surely one of three relations is true:

$$z_1(x) \prec z_2(x), \quad z_2(x) \prec z_1(x), \quad z_1(x) = O\left[z_2(x)\right].$$

In addition it is proved, that a hierarchy of elasticities generates an identical hierarchy of $\mathfrak{L}$–functions [16], i.e. if $E_x(z_1) \prec E_x(z_2)$ then $z_1(x) \prec z_2(x)$. Note that here the relation $\prec$ is interpreted as follows: $z_1(x) \prec z_2(x)$ if and only if $z_1(x) = o[z_2(x)]$. The O-symbol here denotes the asymptotic proportionality of functions: $z_1(x) = O[z_2(x)]$ if and only if $z_1(x) \sim cz_2(x), c > 0$. Further we shall stick to this interpretation of asymptotic notation.

We note additionally several important features of elasticity $\mathfrak{L}$-functions. By (2) for sufficiently small Δx the following approximation is correct:

$$\frac{\Delta z}{z} \approx E_x(z) \frac{\Delta x}{x},$$

which means that $E_x(z)$ is the *coefficient of proportionality between a growth rate variables* z *and* x. Elasticity is a dimensionless quantity, since $E_x(z) = E_{ax}(bz)$ for all constants $a > 0, b > 0$. Of the elasticity inherent properties similar to the properties of logarithm and differentiation operations [17], so the elasticity to logarithmic-exponential functions can be easily calculated.

All these properties allow the use elasticity as a measure computational complexity of algorithm, because elasticity satisfies essential requirements such as *comparability, interpretability* and *computability*. Dimensionless of elasticity and invariance with respect to calculations model is the guarantee comparability: because transition from one model to another is only a polynomial transformation of computational complexity algorithm. Elasticity has a clear interpretation: if $z = z(x)$ is a function computational complexity of algorithm, this means that when value of x (length input data of algorithm) increases by one percent a value of z (execution time of algorithm) will increase by about $E_x(z)$ per cent. Computability also holds: elasticity exists for every $\mathfrak{L}$-function, elasticity of $\mathfrak{L}$-function $z = z(x)$ can always be found directly from (2) or from properties of elasticity. Ease of computation is the main advantage of elasticity in comparison with other measures computational complexity of algorithms.

Elasticity is a local characteristic of function: in general its values change at transition from one value to another value of argument. Meanwhile, for large values of argument in the behavior of elasticity observed certain regularity: different classes of $\mathfrak{L}$-functions (different in growth rate) have fundamentally different behavior of elasticity and on basis asymptotic of elasticity can define a class to which belongs $\mathfrak{L}$-function. Following theorem states that this is true.

Theorem 1 (on classification of $\mathfrak{L}$-functions [16]). *Splitting family of monotonous nondecreasing, "basically positive" $\mathfrak{L}$-functions on the classes Subpoly, Poly, Subexp, Exp, Hyperexp in order of their growth is equivalent a proper splitting according to asymptotic behavior of elasticity these functions at infinity:*

$$Subpoly = \{z(x) \,|\, z(x) \prec e^{O(\ln x)}\} \equiv \{z(x) \,|\, E_x(z) = o(1)\}; \tag{3}$$

$$Poly = \{z(x) \,|\, z(x) = O[e^{O(\ln x)}]\} \equiv \{z(x) \,|\, E_x(z) = O(1)\}; \tag{4}$$

$$Subexp = \{z(x) \,|\, e^{O(\ln x)} \prec z(x) \prec e^{O(x)}\} \equiv \{z(x) \,|\, 1 \prec E_x(z) \prec x\}; \tag{5}$$

$$Exp = \{z(x) \,|\, z(x) = O[e^{O(x)}]\} \equiv \{z(x) \,|\, E_x(z) = O(x)\}; \tag{6}$$

$$Hyperexp = \{z(x) \,|\, e^{O(x)} \prec z(x)\} \equiv \{z(x) \,|\, x \prec E_x(z)\}. \tag{7}$$

A theorem on classification of $\mathfrak{L}$-functions generates five classes complexity of algorithms (subpolinomicial, polynomial, subexponential, exponential and hyperexponential respectively), that fully meet modern classification of algorithms. Properties of elasticity and equivalence (3)–(7) make "transparent" the classes complexity of algorithms, which is extremely important for algorithmic practice.

The classes of complexity are needed primarily to compare algorithms. In one-dimensional theory of complexity for recognition of class to which belongs to an algorithm with a function of $z(x) \in \mathfrak{L}$, we perform following steps: we calculate $E_x(z)$, we find an asymptotic estimate for $E_x(z)$ when $x \to \infty$ and we define class complexity using (3)–(7). It should be noted that hierarchy classes of $\mathfrak{L}$-functions generates the hierarchy classes of complexity algorithms:

$$Subpoly \prec Poly \prec Subexp \prec Exp \prec Hyperexp.$$

Here $K_1 \prec K_2$, where $K_1 \neq K_2$ and $K_1, K_2 \in \{Subpoly, Poly, Subexp, Exp, Hyperexp\}$, means that for any $z_1(x) \in K_1, z_2(x) \in K_2$ always true $z_1(x) \prec z_2(x)$.

Suppose you want to compare algorithms α_1 and α_2 with functions $z_1(x)$ and $z_2(x)$ respectively. First, we need to establish of classes complexity for α_1 and α_2. If these algorithms belong to different classes then hierarchy of these classes defines a relation between α_1 and α_2 in terms the execution time of algorithms. When algorithms α_1 and α_2 belong to same class then much depends on this class:

- if $z_1(x), z_2(x) \in Subpoly, Subexp, Hyperexp$ then for $E_x(z_1) \prec E_x(z_2)$ is always true $z_1(x) \prec z_2(x)$, i.e. algorithm α_1 is asymptotically time-faster than algorithm α_2;
- if $z_1(x), z_2(x) \in Poly, Exp$ then $E_x(z_1) \sim cE_x(z_2), c > 0$. When great length input data of algorithm and $0 < c < 1$ the execution time of algorithm α_1 less approximately $1/c$ times than the execution time of algorithm α_2. When $c > 1$ on the contrary, algorithm α_1 is asymptotically time-slower approximately c times than algorithm α_2. For $c = 1$ is required the analysis in $E_x(z_1), E_x(z_2)$ members of a lower order than constant.

Thus, the use of elasticity can significantly simplify process of comparing algorithms. Because elasticity has simpler form than of the appropriate $\mathfrak{L}$-function, see property 4 of elasticity and

equivalence (3)–(7). The concept of elasticity also extends to functions of many variables. This makes it possible to expand the range of applications of elasticity, for example, use this indicator in analysis of parameterized algorithms.

A private elasticity $E_x(z)$ to function $z = z(x, y)$ respect argument x is elasticity variable z, which is considered a function only of x and at constant y. A private elasticity and private derivative respect argument x associated relation:

$$E_x(z) = z'_x \frac{x}{z} = x(\ln z)'_x. \tag{8}$$

A private elasticity $E_y(z)$ to function $z = z(x, y)$ respect argument y defined as follows:

$$E_y(z) = z'_y \frac{y}{z} = y(\ln z)'_y. \tag{9}$$

Since every $\mathfrak{L}$-function $z = z(x, y)$ is continuous and differentiable in domain where it is defined, then for it in this domain always exist a partial elasticity. Expressions (8), (9) are similar to (2). Consequently, $E_x(z)$ and $E_y(z)$ have all basic properties of elasticity one variable. We mention the most important of these properties and we write them for $E_x(z)$.

1) If $z = z(x, y)$ does not depend on x then $E_x(z) = 0$.

2) A private elasticity is dimensionless quantity: $E_x(z) = E_{ax}(bz)$.

3) A private elasticity of product (of relation) functions $z_1 = z_1(x, y)$ and $z_2 = z_2(x, y)$ is equal to sum (difference) of their private elasticity:

$$E_x(z_1 \cdot z_2) = E_x(z_1) + E_x(z_2), \quad E_x(z_1/z_2) = E_x(z_1) - E_x(z_2).$$

4) A private elasticity for $\mathfrak{L}$-functions of general form $z(x, y) = e^{w(x,y)}$, $w(x, y) \in \mathfrak{L}$, given by:

$$E_x[e^{w(x,y)}] = w(x, y)E_x[w(x, y)].$$

5) A elasticity to function $z = z(x, y)$ respect argument t, where $x = x(t)$ and $y = y(t)$, can express in terms of partial elasticity by formula:

$$E_t(z) = E_x(z)E_t(x) + E_y(z)E_t(y).$$

Let $z = z(n, k)$ is a function complexity for parameterized algorithm, where n is length input data of algorithm and k is parameter of algorithm. After the formal replacement of n by x, of k by y we have $z = z(x, y)$. We assume that $z = z(x, y) \in \mathfrak{L}$. Then $E_x(z)$ is coefficient of proportionality between a growth rate the execution time and the length input data of parameterized algorithm. Similarly, $E_y(z)$ is coefficient of proportionality between a growth rate the execution time and the parameter y.

Two-Dimensional Classification FPT-algorithms Respect to the Complexity

In general, the partial elasticity of $E_x(z), E_y(z)$ are functions that depend on two variables x and y. However, a situation is considerably simplified if we consider the fact that for most of parameterized algorithms the execution time of algorithm is described by $\mathfrak{L}$-function:

$$z(x,y) = q(x) \cdot f(y), \tag{10}$$

where $q(x) \in \mathfrak{L}$ is a quantitative component, and $f(y) \in \mathfrak{L}$ is a parametric component for function $z(x,y)$. The multiplicative form of (10) is typical for algorithms with parametrically dependent cycle that performs an exhaustive search of all possible options with different values of parameter, whereas in body of this cycle performed test the current version and time test depends only on the length of input data algorithm. Method of parametric reduction generates it such algorithms. According to (1), a multiplicative form of (10) for the FPT-algorithms is represented formula:

$$z(x,y) = O(x^{O(1)} \cdot f(y)). \tag{11}$$

Let $z(x,y) = q(x) \cdot f(y) \in \mathfrak{L}$. By properties of elasticity of 1 and 3, partial elasticity of $E_x(z), E_y(z)$ degenerate into conventional elasticity of function one variable:

$$E_x(z) = E_x[q(x) \cdot f(y)] = E_x[q(x)] + E_x[f(y)] = E_x[q(x)], \tag{12}$$

$$E_y(z) = E_y[q(x) \cdot f(y)] = E_y[q(x)] + E_y[f(y)] = E_y[f(y)]. \tag{13}$$

Now $E_x(z)$ depends only on x, and $E_y(z)$ depends only on y. Since $q(x), f(y) \in \mathfrak{L}$ then each of these functions belongs to only one class a *Subpoly, Poly, Subexp, Exp, Hyperexp* respect corresponding argument.

We denote by K_x the class complexity for $\mathfrak{L}$-functions $z(x,y) = q(x) \cdot f(y)$ respect argument x and denote by K_y the class complexity for $z(x,y) = q(x) \cdot f(y)$ respect argument y. Then for every parameterized algorithm with a function complexity of $z(x,y) = q(x) \cdot f(y) \in \mathfrak{L}$ there exist a pair:

$$(K_x, K_y) \in \{Subpoly, Poly, Subexp, Exp, Hyperexp\} \times \{Subpoly, Poly, Subexp, Exp, Hyperexp\},$$

which characterizes the complexity of this algorithm respect a length of input data x and respect value of parameter y. Thus, we arrive at two-dimensional classification parameterized algorithms by respect complexity.

When we rely on two-dimensional approach, then more accurately and a general statement gets the definition of FPT-algorithm and a condition of (11): parameterized algorithm called FPT-algorithm, if the time of its execution represent by function $z(x,y) = q(x) \cdot f(y) \in \mathfrak{L}$, for which is true formula:

$$(K_x, K_y) \in \{Subpoly, Poly\} \times \{Subpoly, Poly, Subexp, Exp, Hyperexp\}.$$

For example, FPT-algorithm for which $(K_x, K_y) = (Poly, Subpoly)$ may has $z = z(x,y)$ of the form:

$$z(x,y) = x^3 \cdot y^{\lambda(\ln y)^{m-1}} = e^{3\ln x} \cdot e^{\lambda(\ln y)^m},\ \lambda > 0,\ 0 < m < 1,$$

because $E_x(z) = 3 = O(1)$ for $x \to \infty$, $E_y(z) = \lambda m(\ln y)^{m-1} = o(1)$ for $\lambda > 0,\ 0 < m < 1$ and $y \to \infty$. The FPT-algorithm with function

$$z(x,y) = (\ln x)x^2 \cdot y^{\lambda(\ln y)^{m-1}} = e^{\ln\ln x} e^{2\ln x} \cdot e^{\lambda(\ln y)^m},\ \lambda > 0,\ m > 1$$

characterized by complexity $(K_x, K_y) = (Poly, Subexp)$, as $E_x(z) = 1/\ln x + 2 = O(1)$ for $x \to \infty,\ E_y(z) = \lambda m(\ln y)^{m-1} = o(y)$ for $\lambda > 0,\ m > 1$ and $y \to \infty$. If

$$z(x,y) = x^5 \cdot e^{\lambda y} = e^{5\ln x} \cdot e^{\lambda y},\ \lambda > 0, \tag{14}$$

then FPT-algorithm has complexity $(K_x, K_y) = (Poly, Exp)$, since $E_x(z) = 5 = O(1)$ for $x \to \infty$, $E_y(z) = \lambda y = O(y)$ for $\lambda > 0$ and $y \to \infty$. The FPT-algorithm with function

$$z(x,y) = \ln x \cdot y^y = e^{\ln\ln x} \cdot e^{y\ln y}$$

has complexity $(K_x, K_y) = (Subpoly, Hyperexp)$, because $E_x(z) = 1/\ln x = o(1)$ for $x \to \infty$ and $E_y(z) = y\ln y(1 + 1/\ln y) = O(y\ln y)$ for $y \to \infty$.

When a parameterized problem is not only FPT, but polynomial-time computable problem, then it has FPT-algorithms whose complexity corresponds to pairs

$$(K_x, K_y) \in \{Subpoly, Poly\} \times \{Subpoly, Poly\}. \tag{15}$$

Such parameterized algorithms are naturally called *polynomial* FPT-*algorithms*. Parameterized algorithms whose complexity corresponds to pairs

$$(K_x, K_y) \in \{Subpoly, Poly\} \times \{Subexp\}, \tag{16}$$

$$(K_x, K_y) \in \{Subpoly, Poly\} \times \{Exp\}, \tag{17}$$

$$(K_x, K_y) \in \{Subpoly, Poly\} \times \{Hyperexp\}, \tag{18}$$

it is expedient to named as *subexponential, exponential* and *hyperexponential* FPT-*algorithms* respectively. Such FPT-algorithms are specific to NP-hard problems.

Two-dimensional classification FPT-algorithms does not contradict concepts that are currently used in parameterized complexity theory as applied to FPT-algorithms, but merely formally clarifies them. For example, today there is a demand for subexponential FPT-algorithms for solving of parameterized versions NP-hard problems [19, 20]. It is considered that for such algorithms of time execution equal to

$$z(x,y) = x^{O(1)} \cdot 2^{o(y)}$$

and satisfies (17). It is obvious, that there always the Cartesian product

$$\{Subpoly, Poly, Subexp, Exp, Hyperexp\} \times \{Subpoly, Poly, Subexp, Exp, Hyperexp\},$$

can be divided into smaller subsets of pairs (K_x, K_y) and, as a consequence, define more detailed classification of parameterized algorithms than (15)–(18).

For trivial parameterization is true $y = x$ and expession (12), (13) becomes:

$$E_x(z) = E_x[q(x) \cdot f(x)] = E_x(q) + E_x(f).$$

In this particular case we arrive at one-dimensional classification of algorithms. A similar case arises when $f(y)$ is identically constant, i.e. $f(y) \equiv c > 0$.

Classification Parameterized Algorithms in Terms Impact of Parameter

For parameterized algorithms of practical interest is the classification, which aims to clarification the level impact parametric component $f(y)$ the multiplicative form (10) on time execution of algorithm. Traditionally, we distinguish between [4, 9]:

- parameterized algorithms *with a weak level of impact* parameter. The time execution such algorithms depends weakly on parametric component $f(y)$. It establishes a predominantly coefficient for $q(x)$;
- parameterized algorithms for which parametric component $f(y)$ and quantitative component $q(x)$ have a *comparable effect*;
- parameterized algorithms *with a dominant level of impact* parametric component. For them, the parameter identifies the main source of complexity. It is these algorithms interesting for parameterized complexity theory.

Unfortunately, in literature on parameterized complexity until are no formal methods of identifying parameterized algorithms with a dominant dependence on parameter. Application for these purposes elasticity gives the positive results for multiplicative forms of presentation functions of complexity and for the family $\mathfrak{L}$-functions.

In fact, a comparison of components $q(x)$ and $f(y)$ is only possible when there is a dependence $y = y(x)$, which describes the relationship between values of parameter and length input data of algorithm. In these circumstances by properties of elasticity 3 and 5, the expression (12) becomes:

$$E_x(z) = E_x[q(x) \cdot f(y)] = E_x[q(x)] + E_x[f(y)] = E_x(q) + E_y(f) \cdot E_x(y). \tag{19}$$

Note that, in (19) the terms $E_x(q)$ and $E_y(f) \cdot E_x(y)$ are identically constant or infinitely small or infinitely large logarithmic-exponential functions, which depend only on x, and because they always are comparable with each other in order of growth rate. We assume that $z(x, y) = q(x) \cdot f(y)$ is not identically constant (case $z(x, y) \equiv c > 0$ is not interesting). We have the following situations.

If when $x \to \infty$ relation

$$E_y(f) \cdot E_x(y) \prec E_x(q) \tag{20}$$

holds then

$$E_x(z) = O[E_x(q) + E_y(f) \cdot E_x(y)] = O[E_x(q)].$$

This means that parameterized component has weak effect on value of $z(x, y) = q(x) \cdot f(y)$. If when $x \to \infty$ relation

$$E_x(q) = O[E_y(f) \cdot E_x(y)] \tag{21}$$

is correct then

$$E_x(z) = O[E_x(q) + E_y(f) \cdot E_x(y)] = O[E_x(q)] = O[E_y(f) \cdot E_x(y)],$$

which indicates a comparable effect of parametric and quantitative components. Finally, if for $x \to \infty$ relation

$$E_x(q) \prec E_y(f) \cdot E_x(y) \tag{22}$$

is true then

$$E_x(z) = O[E_x(q) + E_y(f) \cdot E_x(y)] = O[E_y(f) \cdot E_x(y)].$$

This indicates that parametric component is dominant.

Consequently, we can mathematically rigorously identify a level of impact parametric component $f(y)$ the multiplicative form (10) on time execution of algorithm by examining relations (20)–(22) and using properties of elasticity. The verification process is simplified if we consider

the fact that the dependence $y = y(x)$ usually holds for internal parameter of parameterized algorithm. In this case $y = y(x)$ is a polynomial or subexponential function (in growth rate) and hence $E_x(y) = O(1)$ or $E_x(y) = o(1)$ when $x \to \infty$. If $E_x(y) = O(1)$ then relations (21)–(23) simplifies and takes form:

$$E_y(f) \prec E_x(q),$$

$$E_x(q) = O[E_y(f)],$$

$$E_x(q) \prec E_y(f).$$

Thus, if we have a polynomial dependence of parameter on length of input data, then it all boils down to a comparison of two $\mathfrak{L}$-functions (more precisely to a comparison of their elasticity). When $E_x(y) = o(1)$ we should take into account in (20)–(22) both factors of expression $E_y(f) \cdot E_x(y)$. For the trivial parameterization of relation between complexity classes in (K_x, K_y) identifies the level of impact parameter on time execution of algorithm. If $K_y \prec K_x$ then time execution of parameterized algorithm depends only weakly on parameter. If $K_y = K_x$ then the parametric component and quantitative component have a comparable effect. If $K_x \prec K_y$ then we have a parameterized algorithm with a dominant level of impact of the parameter.

For example, consider the function (14), which is characterized by two-dimensional complexity $(K_x, K_y) = (Poly, Exp)$ and $E_x(q) = 5,\ E_y(f) = \lambda y$. Let $y = \ln x$. Then $E_x(y) = 1/\ln x = o(1)$. Since at $x \to \infty$ holds estimate

$$E_y(f) \cdot E_x(y) = (\lambda \ln x \cdot (1/\ln x)) = \lambda = O(1)$$

then relation (21) is true, i.e. the parametric component and quantitative component have a comparable effect, although there $K_x \prec K_y$. Now let $y = x$. We have a trivial parameterization and $E_x(y) = 1$. The estimate

$$E_y(f) \cdot E_x(y) = \lambda x = O(x)$$

satisfies (22), satisfies relation $K_x \prec K_y$ and shows dominant a level of impact parametric component.

Conclusions

Parameterized complexity theory has received in recent years widespread and is an extremely suitable tool for solving NP-hard problems associated with a number of important applications. It draws our attention to how represent the input data, what data are of interest to problem being solved and what data are sources of the nonpolynomial-time for NP-hard problems. Parameterized complexity theory has given impetus to development of new approaches into designing algorithms (algorithms solving the FPT-problems). At the same time, this theory has generated a wide range of problems. One problem is the lack of simple and convenient tools for analysis of parameterized algorithm where function of complexity depends on many variables.

In this paper we proposed a new indicator of computational complexity for parameterized algorithm, which can be used to measure rate of growth function of complexity from many variables, by which we can to compare the FPT-algorithms for NP-hard problems and to perform analysis level of impact parameter for time execution of the parameterized algorithm. This indicator is the elasticity of function complexity algorithm. In this paper we proposed a two-dimensional classification of parameterized algorithms to multiplicative forms of presentation the functions complexity, we accurately identify the class of FPT-algorithms and we proposed the mathematical method of analysis level of impact parameter for time execution of parameterized

algorithm. The proposed approach to analysis of parameterized algorithms admits further development in terms of increasing a number of parameters, the study of other forms for representing the function complexity.

References

[1] M.Garey, D.Johnson, Computers and intractability: A guide to the theory of NP-completeness, W. H. Freeman and Company, New York, NY, 1979.

[2] G.Ausiello, P.Crescenzi, G.Gambosi, V.Kann, A.Marchetti-Spaccamela, M.Protasi, Complexity and approximation, combinatorial optimization problems and their approximability properties, New York, Springer-Verlag, 1999.

[3] J.Flum, M.Grohe, Parameterized complexity theory, EATCS Texts in Theoretical Computer Science, Springer-Verlag, Berlin, Heidelberg, 2006.

[4] R.Downey, M.Fellow's, Parameterized complexity, New York, Springer-Verlag, 1999.

[5] G.Gottlob, F.Scarcello, M.Sideri, Fixed-parameter complexity in AI and Nonmonotonic reasoning, *Artificial Intelligence*, **138**(2002), 55–86.

[6] H.Bodlaender, R.Downey, M.Fellows, M.Hallett, H.Wareham, Parameterized complexity analysis in computational biology, *Computer Applications in the Biosciences*, **11**(1995), 49–57.

[7] C.Papadimitriou, M.Yannakakis, On the complexity of database queries, *Journal of Computer and System Sciences*, **58**(1999), 407–427.

[8] M.Grohe, The parameterized complexity of database queries, in *Proceedings of the 20th ACM Symposium on Principles of Database Systems*, 2001, 82–92.

[9] R.Niedermeier, Invitation to fixed-parameter Algorithms, *Oxford Lecture series in mathematics and its applications*, Oxford University Press, 2006.

[10] M.Cesati, Compendium of parameterized problems, 2006 (http://bravo.ce.uniroma2.it/ home/ cesati/research/compendium/)

[11] G.Gottlob, R.Pichler, F.Wei, Abduction with bounded treewidth: From theoretical tractability to practically efficient computation, in *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence*, 2008, 1541–1546.

[12] T.Cormen, C.Leiserson, R.Rivest, Introduction to algorithms, The MIT Press, Cambridge, MA, 1990.

[13] S.Baase, A.Gelder, Computer Algorithms: introduction to Design and Analysis, Addison-Wesley, 2000.

[14] V.V.Bykova, Mathematical methods of analysis recursive algorithms, *Journal of Siberian Federal University, Mathematics & Physics*, **1**(2008), no. 3, 236–246.

[15] V.V.Bykova, Complexity and elasticity of the computation, in *Proceedings of the third IASTED International Multi-Conference on Automation, Control, and Information Technology (ACIT-CDA 2010) in cooperation with the Russian Academy of Sciences,* ACTA Press Anaheim-Calgary-Zurich, 2010, 334–340.

[16] V.V.Bykova, Recognition method of algorithms classes on the basis of asymptotics for elasticity functions complexity, *Journal of Siberian Federal University, Mathematics & Physics*, **2** (2009), no. 1, 236–246.

[17] V.V.Bykova, Elasticity of algorithms, *Applied Discrete Mathematics*, **2**(2010), 87–95.

[18] R.Graham, D.Knuth, O.Patashnik, Concrete mathematics, Addison-Wesley, 1994.

[19] L.Cai, D.Juedes, On the existence of sub-exponential time parameterized algorithms, *Journal of Computer and System Sciences*, **67**(2003), 789–807.

[20] J.Flum, M.Grohe, Parameterized complexity and subexponential time, *Bull. Eur. Assoc. Theoretical Computer Sciences EATCS*, **84**(2004), 71–100.

Анализ параметризированных алгоритмов на основе эластичности функций сложности

Валентина В. Быкова

Дан краткий обзор результатов и проблем параметризированной алгоритмики — нового направления теории сложности вычислений. Предложен новый показатель вычислительной сложности параметризированного алгоритма, с помощью которого можно измерять темп роста функции сложности многих переменных. Этим показателем является частная эластичность функции сложности. Предложена двумерная классификация параметризированных алгоритмов для мультипликативной формы представления функций сложности. Математически обоснован метод анализа уровня влияния параметра на время работы параметризированного алгоритма.

Ключевые слова: сложность вычислений, параметризированные алгоритмы, анализ алгоритмов, эластичность алгоритмов.