On Entropy in Computations

Anatol Slissenko
Laboratory for Algorithmics, Complexity and Logic
Université Paris-Est Créteil (UPEC), France
e-mail: slissenko@u-pec.fr

ABSTRACT
This is an attempt to grope for quantitative measures of information structure of runs of algorithms. The prospective goal, likely long-term, is on the one hand, to introduce smoothness and information convergence or information capacity of algorithms with the hope to prove lower bounds on complexity for such algorithms (all practical algorithms intuitively look smooth), and on the other hand, to try to find new ideas for designing efficient algorithms. I start with adapting classical notions of epsilon-entropy of compacts and probabilistic entropy. Though they are not sufficient to reach the goal, they give some ideas how to go further.

Keywords Algorithm, computation, metrics, entropy, information convergence

1. INTRODUCTION

Many concepts widely used in computer science come from mathematics of the pre-computer era. The most famous examples are: logical theory and algorithm (lambda-calculus, recursive function, Turing machine). In hindsight, they played a very important role in the development of theoretical computer science with well visible impact on applications. They stimulated the development of more adequate notions like Boolean circuit and RAM (Random Access Machine), and hence the computational complexity and algorithmics. However, all the mentioned concepts are too wide, and traditional complexity theory is inadequate to the realistic computations. For example, polynomial time (even linear time) is not practical feasibility, and negative algorithmic results (with the existing proofs) say almost nothing relevant to realistic computations (see my remarks on http://lacl.u-pec.fr//slissenko/Miscellaneous/).

Notice that modern mathematics does not study arbitrary functions, nor arbitrary continuous functions, nor even arbitrary smooth functions. It studies particular, always rather smooth, manifolds on which often, though not always, acts a group with some nice properties. An algorithm is not only a function, from the viewpoint of practical computation or from the viewpoint of computational complexity, it is rather a set of runs. This analogy and the geometric nature of many of our intuitive perceptions motivates this attempt to find geometric structure in computations (and later, I hope, in problems).

Measuring similarity between traces (a trace is a concise representation of run as a sequence of events, i.e., of executions of single commands) is the first issue addressed below. As for information convergence, a more or less traditional approach is discussed. It remains not clear whether one can well relate the introduced notions of the flavor of epsilon-entropy [1] and of probabilistic entropy (the situation under consideration is deterministic, so probabilistic treatment is purely technical).

This paper presents: (1) A metric over traces and a measure of global similarity of traces. (2) A notion of information convergence of algorithms. (3) Some very preliminary considerations on measuring structural complexity of some problems.

The notions (1) are intended to approach a formalization of smoothness of algorithms. Those (2) may help to make the just mentioned ones more adequate, and moreover to bridge smoothness with the structural complexity of problems.

2. A METRIC OVER SETS OF TRACES

Similarity of traces. We suppose that an algorithm F (that is fixed below) is a transition system defined *over* a vocabulary of functions (relations are considered as functions) and a finite set of universes that are used to define the semantics. The vocabulary may be infinite. For example, it may contain all constants for rational numbers. The universes are usually infinite. Some functions have fixed interpretations (they are static) and are not changed by algorithms, we assume that standard basic static functions like arithmetical operations and order relations over integers, Boolean bitwise operations, and basic list manipulations are always in the vocabulary. The other functions are dynamic. We distinguish 3 kinds of such functions: input functions V_{inp}, output functions V_{out} and proper internal functions V_{int} used by the algorithm for its work; all these functions constitute V; the nullary functions of V, i.e., of arity zero, correspond to variables of programming languages. The values of input functions are not changed by algorithm, the values of the others are computed by algorithm.

By $\boldsymbol{dom}_n(F)$ we denote the set of inputs of F of length n, and by $\boldsymbol{dom}_n(g)$, where $g \in V$, the set of arguments of g that appear in runs of F for inputs from $\boldsymbol{dom}_n(F)$. We suppose that $\boldsymbol{dom}_n(F) \neq \emptyset$ for all n. By $\boldsymbol{range}_n(f)$ we denote the set of values of f for arguments from $\boldsymbol{dom}_n(f)$, where $f \in V$ or $f = F$. The bound n is usually fixed, and thus, omitted.

We introduce a vocabulary $\widetilde{V}$ of nullary functions obtained from V by replacing each non-nullary function $f : A \to B$ by the set of functions $\{f_a\}_{a \in A}$ with $f_a = f(a)$. We will deal functions from $\widetilde{V}$ for n that is fixed. In particular, V_{int} and V_{out} denote below respectively input and output functions as represented in $\widetilde{V}$.

A *similarity bijection of* V is defined by:
(a) a bijection φ of $\widetilde{V}$ on itself that preserves functions from V_{in} and V_{out};

(b) bijections ψ_g of $\boldsymbol{range}(g)$ onto $\boldsymbol{range}(\varphi(g))$ for $g \in \widetilde{V}$.

Time $\mathbb{T}$ is discrete, i.e., $\mathbb{T} = \mathbb{N}$. A *state* is an interpretation of the vocabulary, and a *run* of F is a sequence of states that starts from the initial state and such that each its non initial state is a result of a transition from the previous one. A transition executes an *event*; each event is either an assignment or the evaluation of a condition followed by changing of control. A run can be represented as a *trace*, i.e., as a sequence of events starting from the initial state. The *causal order* between events is defined as usually.

Notations:
• $\boldsymbol{tr}(x)$ is the *trace corresponding to* input x.
• $|\tau|$ is *the number of events* in trace τ.
• $t^*_F(x)$ and $t_F(n)$ are respectively *time complexity* of F *for a given input* x (i.e.,the number of steps that F uses to process x) and *worst-case time complexity for inputs of the length* $\leq n$ (it is assumed to be the same for inputs of length exactly n).
• $\lambda[x,t]$ is *the value of a function* λ *in the run* for an input x at time instant t; if $t > t^*_F(x)$ then t is replaced by $t^*_F(x)$ in $\lambda[x,t]$.
• $\lambda[x]$ is $\lambda[x, t^*_F(x)]$, i.e., the final value of λ for input x.
• $e(x,t)$ is the event in the run for an input x at an instant t; below when we speak about an event we mean an event of this form, i.e., an *occurrence* of an event in a trace. □

Let Φ be a similarity bijection of V, i.e., a set of mappings mentioned above. Take two traces for two inputs x_k and two time instants t_k, $k = 0, 1$. Suppose that $e(x_k, t_k)$ are assignments $f_k := g_k$, where $f_k, g_k \in \widetilde{V}$. These events are *similar* under Φ if $\varphi(f_0) = f_1$, $\varphi(g_0) = g_1$ and $\varphi_{g_0}\big(g_0[x_0, t_0]\big) = g_1[x_1, t_1]$ (i.e., f_0 and f_1 get similar values). Suppose that $e(x_k, t_k)$ are evaluations of a condition γ_k, where γ_k is a Boolean function from $\widetilde{V}$. Denote the value of this evaluation by v_k. These two evaluations are *similar* under Φ if $\varphi_{\gamma_0}(v_0) = v_1$, i.e., v_0 is similar to v_1, and the *next* events determined by the control produced by these evaluations are also similar.

Let x_k, $k = 0, 1$, be two inputs. Let $\mathcal{S}_k$ be a set of disjoint sequences of events from $\boldsymbol{tr}(x_k)$, $|\mathcal{S}_k| = s \geq 1$ for $k = 0, 1$.

By a *similarity bijection* of these 2 sets we mean the following: a set $\mathcal{B}$ of s similarity bijections B_{η_0} of V attributed to sequences η_0 of $\mathcal{S}_0$, a bijection π from $\mathcal{S}_0$ onto $\mathcal{S}_1$ (π establishes a one-to-one correspondence between sequences of $\mathcal{S}_0$ and $\mathcal{S}_1$), and a set $\mathcal{H}$ of s bijections h_{η_0}, $\eta_0 \in \mathcal{S}_0$, such that each bijection h_{η_0} is from η_0 onto $\pi(\eta_0)$, it preserves the similarity of events with respect to B_{η_0} and the causal order between all events of all sequences (i.e., if $e \in \eta_0 \in \mathcal{S}_0$, $e' \in \eta'_0 \in \mathcal{S}_0$ and e causally precedes e' in $\boldsymbol{tr}(x_0)$ then $h_{\eta_0}(e)$ causally precedes $h_{\eta'_0}(e')$ in $\boldsymbol{tr}(x_1)$). We call such a bijection a *similarity bijection* from $\mathcal{S}_0$ to $\mathcal{S}_1$. If such a bijection exists then $\mathcal{S}_0$ and $\mathcal{S}_1$ are called *similar*. Notice that there may be several similarity bijections from $\mathcal{S}_0$ to $\mathcal{S}_1$.

A measure of similarity. We can measure similarity between 2 traces $\tau_k =_{df} \boldsymbol{tr}(x_k)$, $k = 0, 1$, by choosing in each τ_k a set $\mathcal{S}_k$ of disjoint subsequences of events such that $\mathcal{S}_0$ and $\mathcal{S}_1$ are similar, and counting the number of non-similar events. Let $\mathcal{S}_0$ and $\mathcal{S}_1$ be two such sets. Denote the similarity bijection from $\mathcal{S}_0$ to $\mathcal{S}_1$ by β_{01} (it represents all involved mappings, and will be used informally); we call such β_{01} a *similarity* between $\mathcal{S}_0$ and $\mathcal{S}_1$.

Set $d_{\beta_{01}}(\tau_0, \tau_1) = |\tau_0| + |\tau_1| - 2\big|\bigcup \mathcal{S}_k\big| + 2(s-1)$, where $\big|\bigcup \mathcal{S}_k\big|$ is the number of events in all sequences of $\mathcal{S}_k$ (k may be any as $\big|\bigcup \mathcal{S}_0\big| = \big|\bigcup \mathcal{S}_1\big|$). One can see that $|\tau_0| + |\tau_1| - 2\big|\bigcup \mathcal{S}_k\big|$ is the cardinality of a symmetric difference between τ_0 and τ_1 if one identifies similar events from $\mathcal{S}_0$ and $\mathcal{S}_1$. The term $2(s-1)$, that is 0 if we take only one sequence in each trace, is 'payment' for each extra subsequence that one wishes to use. It is added in order to avoid 'very good' trivial similarity by very many very short subsequences, e.g., singletons (i.e., by one command executions).

Distance between traces is defined as

$d(\tau_0, \tau_1) = \min\{d_{\beta_{01}}(\tau_0, \tau_1) : \text{over all similarities } \beta_{01}\}$.

As the usefulness of the case of $s > 1$ is not yet evident, *below we assume that* $s = 1$, i.e., only one similarity bijection is permitted to compare traces.

Clearly, $d(\tau_0, \tau_0) = 0$ and $d(\tau_0, \tau_1) = d(\tau_1, \tau_0)$. The equality $d(\tau_0, \tau_1) = 0$ does not mean that $\tau_0 = \tau_1$, it means, however, that these traces are completely similar. One can see that *the relation* $d(\tau_0, \tau_1) = 0$ *between traces is an equivalence.*

We show that d *is a semi-metric on traces or a metric on classes of equivalent traces.* (A function ρ is a semi-metric if $\rho(x,x) = 0$, $\rho(x,y) = \rho(y,x)$, $\rho(x,y) \leq \rho(x,z) + \rho(z,y)$, but not necessarily $\rho(x,y) = 0 \Rightarrow x = y$). It suffices to prove the triangle inequality.

Let $\tau_k = \boldsymbol{tr}(x_k)$, $k = 0, 1, 2$, be 3 traces, and let similarities β_{km}, $km \in \{02, 01, 12\}$ (here km means a pair (k, m)), be such that $d_{\beta_{km}}(\tau_k, \tau_m) = d(\tau_k, \tau_m)$, and let the respective subsequences of events of traces τ_k and τ_m for β_{km} be respectively σ_{km} and σ_{mk}. In our case of $s = 1$ we have $\mathcal{S}_{km} = \{\sigma_{km}\}$ and thus $\big|\bigcup \mathcal{S}_{km}\big| = |\sigma_{km}|$. The triangle inequality is:

$$d(\tau_0, \tau_2) \leq d(\tau_0, \tau_1) + d(\tau_1, \tau_2) \ (*).$$

As $d(\tau_k, \tau_m) = |\tau_k| + |\tau_m| - 2|\sigma_{km}|$, formula $(*)$ can be rewritten as $|\tau_0| + |\tau_2| - 2|\sigma_{02}| \leq |\tau_0| + |\tau_1| - 2|\sigma_{01}| + |\tau_1| + |\tau_2| - 2|\sigma_{12}|$, that is equivalent to

$$|\sigma_{01}| + |\sigma_{12}| \leq |\tau_1| + |\sigma_{02}| \quad (**)$$

Look at relative positions of $\beta_{01}(\sigma_{01})$ and of σ_{12} in τ_1 (here and below we mean by β_{01} the appropriate mapping from β_{01}, i.e., the mapping that was denoted h_{η_0} above, and that, in the context under consideration, is $h_{\sigma_{01}}$). If these subsequences of τ_1 are pairwise disjoint then $|\sigma_{01}| + |\sigma_{12}| \leq |\tau_1|$, as they all lie inside τ_1. Hence, in this case we have $(**)$.

Suppose that these subsequences intersect. For $\sigma = \beta_{01}(\sigma_{01}) \cap \sigma_{12}$ there holds $|\sigma_{01}| + |\sigma_{12}| \leq |\tau_1| + |\sigma|$. Subsequence $(\beta_{01})^{-1}(\sigma)$ of τ_0 and subsequence $\beta_{12}(\sigma)$ of τ_2 are similar: $\beta_{12}\big(\beta_{01}(\beta_{01}^{-1}(\sigma))\big) = \beta_{12}(\sigma)$. As σ_{02} is chosen as the longest similar subsequence of τ_0 and of τ_2 (for some bijection) then $|\sigma| \leq |\sigma_{02}|$, and hence $(**)$.

The set of all traces for inputs of a given length n, denote it $\boldsymbol{Tr}_n$, is a finite, and thus compact set. If one uses a metric $\frac{d}{2 \cdot t_F(n)}$ then the diameter of $\boldsymbol{Tr}_n$ becomes ≤ 1. In any case we can look at minimal epsilon-nets in $\boldsymbol{Tr}_n$ (recall that a subset S of a compact is an *vep*-net if the closed balls of radius ε centered at the points of S cover the compact), and evaluate whether epsilon-entropy (the base two logarithm of the size of a minimal epsilon-net [1]) permits to compare algorithms from the viewpoint of the structural complexity of the sets of their traces. Below we give give 2 examples that outline some issues concerning ε-entropy in compact spaces of traces.

We take the function d as defined above, not devised by the time complexity, so we take as epsilon a natural number.

Example: Palindrome recognition. Consider a trivial palindrome recognition of strings of even positive length over $\mathbb{B} = \{0, 1\}$. The vocabulary consists of alphabets $\mathbb{B}$, $\{\boldsymbol{true}, \boldsymbol{false}\}$, $\mathbb{N}$ and functions $eq : \mathbb{B}\times\mathbb{B} \to \mathbb{B}$ (it checks the equality of characters of $\mathbb{B}$), $+1 : \mathbb{N} \to \mathbb{N}$, $i :\to \mathbb{N}$ (loop counter), input function $w : \mathbb{N} \to \mathbb{B}$ ($w(i)$ is the ith character of the input string, $i \geq 1$), input function $n :\to \mathbb{N}$ (the length of the input), output function $r :\to \mathbb{B}$ ($r = 1$ means that the input is palindrome, otherwise – non palindrome).

The first algorithm, denote it G, processes its input by evaluating $eq(w(i), w(n - i + 1)$ for $1 \leq i \leq \frac{n}{2}$, starting with $i = 1$ and advancing i as $i := i + 1$ if the analyzed characters are equal. If the algorithm arrives at $eq(w(\frac{n}{2}), w(\frac{n}{2} + 1))$, it outputs 1 otherwise it outputs 0. The initial values: $i = 1$, $r = 1$. Below we use not rigorous notations, namely, sometimes we write directly the values of terms (though for similarity we should see the terms). The algorithm has, roughly speaking, two types of traces (the details depend on the level of abstraction we use). For assignments $i := i+1$ we write in the right-hand term the value of i (instead of character i) and in brackets we write the acquired value. Similar for condition $i < \frac{n}{2}$.

Type 1. Non palindrome (for each type of trace there are inputs w that give such traces):
$A(w, 1) =_{df} \big(\neg eq(w(1), w(n)),\ r := 0\big)$
and for $m > 1$:
$A(w, m) =_{df}$
$\big(eq(w(1), w(n)),\ i[= 1] < \frac{n}{2},\ i := 1 + 1[= 2], \ldots,$
$eq(w(m - 1), w(n - m + 2)),\ i[=]m - 1 < \frac{n}{2},$
$i := m - 1 + 1[= m],\ \neg eq(w(m), w(n - m + 1)),\ r := 0\big)$
for $1 < m \leq \frac{n}{2}$.

Type 2. Palindrome:
$B(w) =_{df}$
$\big(eq(w(1), w(n)\big),\ i[= 1] < \frac{n}{2},\ i := 1 + 1[= 2], \ldots,$
$eq(w(\frac{n}{2}),\ w(\frac{n}{2} + 1),\ \neg(i[= \frac{n}{2}] < \frac{n}{2}),\ r := 1\big)$.

As the similarity does not demand similarity of the values of arguments, we may take here trivial identical bijection to get a good estimation of the entropy. One see that there is only $n/2$ classes of equivalence of traces, that is for inputs inputs W and W' with the respective traces $A(W, m)$ and $A(W', m)$ or $B(W)$ and $B(W')$: $d(A(W, m), A(W', m)) = 0$ and $d(B(W), B(W')) = 0$. The non-trivial relations for appropriate inputs:
$d(A(W, m), A(W', m + 1)) = 3$,
$d(A(W, m), B(W')) = 3(\frac{n}{2} - m) + 2$, $1 \leq m \leq (\frac{n}{2} - 1)$.

So there are non trivial ε-nets for $\varepsilon \geq 3$, and their size is $c \cdot n$, where $c \leq \frac{1}{2}$ is a constant depending only on ε (we do not take into account small additive constant). Thus log of the size, i.e., ε-entropy of the set of traces, is $\log n - c_0$ for some $c_0 > 0$.

Now take another algorithm G' for palindrome recognition; its vocabulary is slightly bigger and is clear from the context. For simplicity consider only inputs whose length is positive and is divisible by 4. It applies two different procedures G_0 and G_1 depending on the first bit of the input. If this bit is 0, it applies G_0 that is like G, otherwise it applies G_1 that works as follows. The procedure G_1 uses 2 loop counters: i that goes from 1 to $\frac{n}{4}$, and a counter j that goes from $\frac{n}{2}$ to $\frac{n}{4} + 1$. In the loop with i it works as G. In the loop with j it calculates $(w(j) \oplus w(n - j + 1))$, where $\oplus$ is sum mod 2, and treat 0 as $\boldsymbol{true}$, and 1 as $\boldsymbol{false}$. And G_1 alternates these two loops: one eq-comparison, one $\oplus$-comparison, one eq-comparison etc.

The set $\mathcal{G}_0$ of traces produced by G_0 has a metric structure similar to that of G. The set $\mathcal{G}_1$ of traces of G_1 has also a metric structure that is alike from the viewpoint of the number of types of traces and their relative distances.

Consider $\tau_k \in \mathcal{G}_k$ with the same number m of comparisons. The distance between τ_0 and τ_1 is $> \frac{m}{2}$. Thus, ε-nets in the sets of traces of G' are roughly twice bigger than for G. Algorithm G' is 'worse' than G, however its epsilon-entropy is slightly bigger because its set of traces is more diverse. This difference is in the additive constant c_1 in the formula for epsilon-entropy $\log n + c_1$, this time $c_1 > 0$. □

Example: Convolution of bit vectors. We consider the straightforward algorithm that follows the definition of the discrete convolution. Input data alphabet is $\mathbb{B} = \{0, 1\} \subset \mathbb{N}$, and output and intermediate data alphabet is $\mathbb{N}$. Input functions are $n :\to \mathbb{N}$ (the length of each of two input vectors), $x, y : \mathbb{N} \to \mathbb{B}$ (two input bit vectors), output function $z : \mathbb{N} \to \mathbb{N}$ (output vector with $(2n-1)$ components of natural numbers, numbered from 0 to $(2n - 2)$), internal functions $\chi : \mathbb{N} \times \mathbb{N} \to \mathbb{B}$ (products of input bits), $\zeta : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ (partial sums in convolution calculation), $i, j, k, s, p, q :\to \mathbb{N}$ (these are loop counters, and thus functions, and for more flexibility in bijections it is better to have them different for different loops), addition and multiplication of natural numbers. If we describe domains and ranges of z, χ, ζ more precisely, that may facilitate more exact analysis of the structure of computations. For example, we may set $\boldsymbol{range}(z(n - 1)) = \{0, 1, \ldots, n\}$.

In order to simplify the presentation of the problem, assume that $x(i) = y(i) = 0$ for $n - 1 < i \leq (2n - 2)$. Then the function to compute consists of $(2n - 1)$ components $z(k) = \sum_{i=0}^{i=k} x(i)y(k - i)$, $0 \leq k \leq (2n - 2)$. The algorithm (a value of n is taken as one of the inputs) works as follows: set $x(p) = y(p) := 0$ in a loop $n - 1 < p \leq (2n - 2)$ (strictly speaking one cannot change inputs but on can simulate the mentioned operation); $\chi(i, j) := x(i) \cdot y(j)$ are computed in a double loop $0 \leq i, j \leq (2n - 2)$; $\zeta(k, 0) := \chi(k, 0)$; $\zeta(k, p) := \zeta(k, p - 1) + \chi(p, k - p)$ for $0 < p \leq k \leq (2n - 2)$; $z(q) := \zeta(q, q)$ for $0 \leq q \leq (2n - 2)$. The inequalities $0 \leq z(k) \leq (k + 1)$ for $0 \leq k \leq (n - 1)$ and $0 \leq z(k) \leq (2n - k - 1)$ for $n \leq k \leq (2n - 2)$ give the maximal values of natural numbers that can appear in the calculations by formulas above. We assume that the algorithm executes the arithmetic operations as atomic ones, so we do not go down to bitwise operations.

We compare traces within identical bijections; suppose there are no others to take into account (very likely, it is true though I had no time to prove it rigorously).

Let x and y be 2 inputs of length n. Denote by K_x and K_y the number of 1 in respectively x and y, and set $U(x, y) =_{df} \{(i, j) : x(i) = y(j) = 1\}$. Clearly, $z[x, y](k) = |U(x, y) \cap \{(i, j) : i + j = k\}|$, and $\sum_k z[x, y](k) = |U(x, y)| = K_x \cdot K_y$.

Suppose that $x(p) = 1$ and $x(q) = 0$ for some $0 \leq p, q \leq (n - 1)$, $p \neq q$. Take x' that coincides with x everywhere except these two positions: $x'(p) = 0$ and $x'(q) = 1$. We have $|\{m : y(m - p) = 1\}| = K_y$ and $|U(x, y)| = |U(x', y)|$ but $(p, m - p) \notin U(x', y)$.
For m, $0 \leq m \leq (2n - 2)$, such that $y(m - p) = 1$ compare
$z[x, y](m) = |U(x, y) \cap \{(i, j) : i + j = m\}| =$
$|U(x, y) \cap \{(p, m - p)\} \cap \{(i, j) : i + j = m \wedge i \neq p\}|$ and
$z[x', y](m) = |U(x', y) \cap \{(i, j) : i + j = m\}|$. These values are different: $z[x, y](m) \neq z[x', y](m)$. Thus, the d-distance between $\boldsymbol{tr}(x, y)$ and $\boldsymbol{tr}(x', y)$ is $\geq 2K_y$ (if we take into account the number of different intermedi-

ate results then the lower bound will be bigger).

Suppose that $K_x = K_y = \frac{n}{2}$, and for simplicity assume that n is even. Then the number of inputs with the mentioned number of 1's is $\binom{n}{n/2}^2 \approx \frac{2^{n+1/2}}{\sqrt{\pi n}}$. Thus. there are at least exponential number of traces (roughly, at least 2^n) with pairwise distances not smaller than n. Hence, ε-entropy for ε of order smaller than n is at least n, i.e., exponentially bigger than in the case of palindrome recognition though the time complexity of the convolution algorithm is only quadratic as compared the linear complexity of palindrome recognition.
The latter remark only underlines that a high structural diversity of traces does not mean a high time complexity. Quite the contrary, a higher structural complexity may correspond to a lower time complexity. □

Remark on auto-similarity. If we ask ourselves what is the difference in the structure of executions of diagonal algorithms and of executions of practical algorithms, we can see that the behavior of a diagonal algorithm strongly varies from input to input. On the other hand, for practical algorithms, I dare to say even for any practical algorithm, its behavior on any long input is similar to its behavior on a fixed amount of rather short inputs. This observation suggests that we can try to formalize auto-similarity as a possibility to cover long runs by runs from a fixed finite set. Such a viewpoint can be implemented along the following lines.

Let $\mathcal{B}$ be a finite set of traces of a given algorithm F. The set $\mathcal{B}$ *covers* all traces of F if the following holds for any trace τ of F. There exists a (finite) set of pairs (τ_i, h_i) such that:
• $\tau_i \in \mathcal{B}$, h_i is a similarity bijection of τ_i into τ,
• any $e \in \tau$ is covered, i.e., $e = h_i(e')$ for some i and $e' \in \tau_i$,

The size of $\mathcal{B}$ is a quantitative characteristic of auto-similarity.

Whether such an approach is productive or not, needs a study. One can modify it taking approximate coverage. It would be interesting to find an appropriate approach to auto-similarity in terms of metric spaces.

3. ON INFORMATION CONVERGENCE OF COMPUTATIONS

Here we consider an algorithm F (this framework can be applied to rather general sets of sequences of terms) that is fixed as well as the bound n on the length of inputs; the latter is presumed but omitted in the notations below. We use our previous setting.

Notations. For a set of functions $\Lambda \subseteq V$ and a mapping T from Λ into $\mathbb{T}$ and an input x:
• $\Lambda[x, T]$, is the set of values $\lambda[x, T(\lambda)]$ for $\lambda \in \Lambda$.
• $\Lambda[x]$ is $\Lambda[x, t^*_F]$, i.e., the set of final values of Λ.
• $\boldsymbol{range}(\lambda)$, where $\lambda \in V$, is the set of values of λ for inputs from $\boldsymbol{dom}(F)$.
• $\boldsymbol{range}(\Lambda)$ is the set of sets $\boldsymbol{range}(\lambda)$ for $\lambda \in \Lambda$.
• $R(\Lambda, Z, T) =_{df}$
$\{x \in \boldsymbol{dom}(F) : \forall\, \lambda \in \Lambda\ \lambda[x, T(\lambda)] = Z(\lambda)\}$, where Z is a mapping from Λ into respective ranges, i.e., $Z(\lambda) \in \boldsymbol{range}(\lambda)$ for $\lambda \in \Lambda$. In other words, $R(\Lambda, Z, T)$ is the set of inputs, taken over all traces, for which a function λ has value $Z(\lambda)$ at instant $T(\lambda)$
• $R(\Lambda, Z)$ is $R(\Lambda, Z, T)$ with $T(\lambda) = t_F(n)$, i.e. $R(\Lambda, Z)$ is the set of inputs for which the final value of any $\lambda \in \Lambda$ is $Z(\lambda)$.
• $\overline{R}(\Lambda, Z) =_{df} (\boldsymbol{dom}(F) \setminus R(\Lambda, Z))$; similar for $\overline{R}(\Lambda, Z, T)$. I.e., $\overline{R}_n(\Lambda, Z)$ is the complement of $R(\Lambda, Z)$.
• $M =_{df} \left|\boldsymbol{range}(F)\right|$ is the number of outputs for inputs from $\boldsymbol{dom}(F)$. □

Now we proceed in a more or less traditional way using probabilistic terminology. The situation under study is quite deterministic. The probability theory does not need the notion of randomness to be applied; it needs just a probability measure.

The sets $R(V_{out}, Z)$ for $Z(\lambda) \in \boldsymbol{range}(\lambda)$, $\lambda \in V_{out}$, constitute a partition of $\boldsymbol{dom}(F)$ into M subsets. We consider that each of these sets has a uniform probabilistic distribution with total probability $\frac{1}{M}$; denote the probability measure defined by this distribution by $\mathbb{P}$. Notice that the set $R(V_{out}, Z)$ may contain one input, and in this case the measure of this input is $\frac{1}{M}$; the set $R(V_{out}, Z)$ may contain several elements, say, K ones, and in this case the measure of each element of this set is $\frac{1}{K \cdot M}$.

We wish to formalize the following intuition. At any given instant the algorithm sees only very limited piece of information. At the beginning of its work algorithm F knows nothing about its input, and hence about its result, and thus its entropy (informally, the uncertainty of its knowledge about the result) is maximal. At the end of its work the algorithm knows all about the result for its input, and hence, its entropy is zero. We represent the entropy concerning the initial knowledge about what result may appear in the classical way as $-M\left(\frac{1}{M}\log\frac{1}{M}\right) = \log M$. In the process of its work the entropy goes to zero, and the speed of this convergence says something about the quality of processing the data. Now we try to do it formally.

The formalization of the intuition sketched above is not so evident. We may reason in terms of the values of functions from $\widetilde{V}$, and ask how the knowledge of these values affect the knowledge about the result. Imagine that the algorithm has read the input. If the knowledge about the input is disposable then there is no obvious way to avoid the conclusion that the result is known. A remedy that immediately come to mind is to restrict the values under scrutiny to pieces whose bit length is much smaller than the length of inputs. In the example of convolution above we permitted to put up to $\log n$ bits in a register. However, even $\log n$ does not look adequate. Indeed, suppose that an algorithm for convolution simply reads $\log n$ bits of its input and assigns this string as a value of some function (variable) without any calculations. Then this value, taken as it is, gives not less information about the result than the corresponding calculations with these bits. It is better to take a constant number of bits. In this case, one can consider that they give not less of information than the calculations with them because the algorithm may have a pre-calculated table of these results of calculations. Another way to deal with values containing non-constant (e.g., $\log n$) number of bits, is to choose these values with clear understanding of its semantics. If we analyze a particular algorithm, it looks reasonable. If we wish to say something about any algorithm for a given problem, we should be cautious. Below, we tacitly mean that we analyze a concrete algorithm, and our choice of values to reason about is well done.

We start with a 'local'question: how the information in chosen values of some functions influences the knowledge about a particular value of a particular piece of the result. It may be useful to look at the question from viewpoint of partitions of measurable spaces. Consider the space $\boldsymbol{dom}(F)$ with measure $\mathbb{P}$. This measure

has the same value on any set of partition
$\zeta =_{df} \{F^{-1}(y) : y \in \boldsymbol{range}(F)\}$. Ascribe to each its point, i.e., to each input, x the values $F(x)$. This value may be a vector as in the case of convolution. The information function [2], 2.2, of ζ is
$I(\zeta)(x) = -\sum_{A\in\zeta} \mathbf{1}_A(x) \log \mathbb{P}(A)$, where $\mathbf{1}_A$ is a characteristic function of the set A, i.e., $\mathbf{1}_A(x) = 1$ if and only if $x \in A$. The entropy is the average (the expectation) of the information function:
$H(\zeta) = -\sum_{A\in\zeta} \mathbb{P}(A) \log \mathbb{P}(A)$. For our case this gives $\log M$ as mentioned above. Below we consider other partitions based on ζ.

Let Λ be a set of functions (intuitively it should be 'small'). Let Z be the values of Λ at some T; there can be many inputs x with traces where $\Lambda[x, T] = Z$. Take a set $\Gamma \subseteq V_{out}$ and a set U of values from the corresponding ranges: $U(\gamma) \in \boldsymbol{range}(\gamma)$ for $\gamma \in \Gamma$.

How does the knowledge that $\Lambda[x, T] = Z$ 'approximate' the knowledge about U?

The partition of $\boldsymbol{dom}(F)$ that describes the relevant information about Γ and U consists of $R(\Gamma, U)$ and its complement $\overline{R}(\Gamma, U)$. The partition that describes the information corresponding to $\Lambda[x, T] = Z$ consists of $R(\Lambda, Z, T)$ and its complement. We ask what is the information about $\Gamma = U$ (that is about $R(\Gamma, U)$) under the condition $\Lambda[x, T] = Z$. Hence, we look at the partition of $R(\Lambda, Z, T)$ into $R(\Lambda, Z, T) \cap R(\Gamma, U)$ and $R(\Lambda, Z, T) \cap \overline{R}(\Gamma, U)$. The probability measures of these 2 sets are respectively conditional probabilities that can be informally written as
$p(T) =_{df} \mathbb{P}(\Gamma = U \big| \Lambda[T] = Z)$ and $\mathbb{P}(\Gamma \neq U \big| \Lambda[T] = Z)$. These probabilities define the entropy of the partition consisting of 2 sets:
$h_0(T) =_{df} -p(T) \log p(T) - (1-p(T)) \log(1-p(T))$. Take as T the mapping $T(\lambda) = t$ for some $t \in \mathbb{T}$ for all $\lambda \in \Lambda$. Observing the value of $h_0(t)$ when t goes to $t_F(n)$ we see the convergence of information about $\Gamma = U$ given by $\Lambda[t] = Z$ during the execution of F. We can also look at the convergence of information of any nature defined in terms of partitions.

On the basis on the 'local' analysis of information convergence described above one can put 'global' questions about worst-case or average convergence.
It seems also to be interesting to find a productive view of algorithm executions as dynamic systems over partitions of measurable spaces.

Example: Palindrome recognition. We discuss algorithm G for which we have $M = 2$. The cardinality of the set $R(r, 1)$ of palindromes is $2^{\frac{n}{2}}$, and the cardinality of the set $R(r, 0)$ of non-palindromes is $(2^n - 2^{\frac{n}{2}})$. In spite of this difference of sizes these sets have the same measure: $\mathbb{P}(R(r, 0)) = \mathbb{P}(R(r, 1)) = \frac{1}{2}$. Consider an instant T where the value of i is k, $1 \leq k \leq \frac{n}{2}$ and $eq(w(k), w(n - k + 1))$ is evaluated to $\boldsymbol{v}$.

Suppose $\boldsymbol{v} = \boldsymbol{false}$ (trivial case). In this case
$\mathbb{P}(R(r, 1) \cap R(eq, \boldsymbol{false}, T)) = \mathbb{P}(\emptyset) = 0$,
$\mathbb{P}(\overline{R}(r, 1) \cap R(eq, \boldsymbol{false}, T)) =$
$\mathbb{P}(R(r, 0) \cap R(eq, \boldsymbol{false}, T)) = \mathbb{P}(R(eq, \boldsymbol{false}, T))$, and hence $\mathbb{P}(r = 1 | eq = \boldsymbol{false}) = 0$ and
$\mathbb{P}(r = 0 | eq = \boldsymbol{false}) = 1$, and thus the respective entropy is 0, i.e., the algorithm knows all.

Suppose $\boldsymbol{v} = \boldsymbol{true}$. In this case
$\mathbb{P}(R(r, 1) \cap R(eq, \boldsymbol{true}, T)) = \mathbb{P}(R(r, 1)) = \frac{1}{2}$,
$\mathbb{P}(R(eq, \boldsymbol{true}, T)) = \mathbb{P}(R(r, 1)) + \mathbb{P}(\{w : w(1..k) =$
$w(n..(n-k+1)) \wedge w((k+1)..\frac{n}{2}) \neq w((n-k)..\frac{n}{2}+1)\}) =$
$\frac{1}{2} + \frac{1}{2^n - 2^{\frac{n}{2}}} \cdot 2^k \cdot (2^{n-2k} - 2^{\frac{n-2k}{2}}) = \frac{1}{2} + \frac{2^{n-k} - 2^{\frac{n}{2}}}{2^n - 2^{\frac{n}{2}}}$. One can see that for $k = \frac{n}{2}$ the latter value is $\frac{1}{2}$ and thus, $\mathbb{P}(r = 1 | eq = \boldsymbol{true}) = 1$, and the entropy becomes 0. For $k = 1$ the value is close to 1 and thus, $\mathbb{P}(r = 1 | eq = \boldsymbol{true})$ is close to $\frac{1}{2}$, and the entropy remains close to its maximal value 1. □

Example: Convolution of bit vectors. It is intuitively obvious that the usual algorithm used in the previous section ensures a certain information convergence from the very beginning of its execution. This is less obvious for algorithms based Fourier Transform (FT). We put aside what algorithm for FT is used, and just notice that even the result of FT of one multiplicand, say x (we use the notations of the example in the previous section), give some 'advanced' information about the result.
We do not enter into technical details. The first component of the FT of x is simply K_x. Clearly, the knowledge of K_x diminishes the set of inputs that might give this or that result, and thus diminishes the appropriate entropy. For example, suppose $K_x = 0$. Then the result is known, it is $z(k) = 0$ for all k. Suppose now $K_x = n$ (i.e., take the maximal value). Then, as in the previous case, there is only one x with this property, and again we get more information about possible results. Similar for other values of K_x. The combinatorics that gives good approximations to the changes of entropy is very laborious. □

Remark. In both examples the estimations of measure describing the information are expressions with the exponentiation. For example, for palindrome recognition the information convergence is determined, in an evident way, by 2^{n-k} (that will be in the denominator in the expression of the probability we are calculating, and hence of the entropy). And these exponential functions reappear in formulas for entropy. Il seems to be simpler and more productive to evaluate the speed of convergence in terms of information, that is in terms of logarithm of these exponential expressions. Taking appropriate approximations we arrive at polynomials that are much more comprehensible.

The measures introduced in this and previous sections probably permit to compare different algorithms not only by comparing its characteristics like this or that entropy, or information, but by introducing similarity between their sets of traces.

The notions introduced above deal only with traces for a given input length. It might be useful to find notions similar in spirit for the set of all traces for all inputs. □

4. TOWARDS METRICAL ANALYSIS OF PROBLEMS STRUCTURE

The discourse in the previous sections gives some ideas about metrical analysis of the structure of computations. However the prospective goal is to arrive to technical concepts apt to information analysis of problems. The basic difficulty is that such concept cannot be on the one hand, unrelated to the concept of algorithm, and on the other hand, cannot be too algorithmic in order to be applicable to any algorithm. Here I can describe one preliminary observation concerning the structure often given by the definition of a problem.

Consider 3 examples: palindrome recognition, convolution and SAT (the problem of satisfiability of propositional formulas). In palindrome recognition we look for the longest prefix of an input string whose inverse is also a suffix of this string, and then the result is determined

by the length of this prefix compared to the length of the input. So we look for a rather simple substructure of input. Similar for the other examples.

In convolution we look for each k, $0 \leq k \leq (2n-2)$, for the set $\{(i,j) : x(i) = y(j) = 1 \wedge i + j = k\}$, and take its cardinality as the kth output.

In SAT we look for the longest truth values assignment that does not make the input formula 'trivially' false. We call such an assignment a *maximal sat-assignment*. By "does not make the input formula 'trivially' false" we mean the following. Given a partial assignment we put its truth values into the formula, and then apply to the obtained formula with constants standard simplifications for constants and repetitions (one can say that we apply all standard simplifications that do not augment the length of the formula). If the obtained formula is not the constant $\boldsymbol{true}$ then we say that the assignment does not make the formula trivially false. For CNF this corresponds, modulo evident detail, to the calculation of the number of disjuncts satisfied by the assignment.

Consider SAT in more detail. Let Φ be a propositional formula, $\boldsymbol{Var}(\Phi)$ be the set of its propositional variables, and $\boldsymbol{Lit}(\Phi)$ be the set of its literals (a literal is a variable or its negation). We assume that Φ is constructed from literals with the help of $\wedge$ and $\vee$. A *partial assignment* can be represented as a subset of literals that contains no variable and its negation; a literal in this set represents its assignment to $\boldsymbol{true}$. For an assignment A denote by $\Phi(A)$ the *residue* formula obtained by the simplifications mentioned just above.

For 2 formulas Φ_0 and Φ_1 with the same number of literals denote $\beta(\Phi_0, \Phi_1)$ the number obtained in the following manner. Take a bijection $h : \boldsymbol{Lit}(\Phi_0) \to \boldsymbol{Lit}(\Phi_1)$ that commutes with the negation: $h(\overline{v}) = \overline{h(v)}$. In Φ_1 and $h(\Phi_0)$ (this is the formula obtained from Φ_0 by replacing each its literal by its h-image) throw away equal subformulas. Denote the sum of sizes of the remaining 2 formulas by $\beta_h(\Phi_0, \Phi_1)$. The minimum of $\beta_h(\Phi_0, \Phi_1)$ over h is $\beta(\Phi_0, \Phi_1)$. For formulas with the same number of variables the function β is a semi-metric.

One can see that measure is also applicable to formulas with different numbers of variables. In the latter case h is chosen as a bijection from the smaller set of literals to the bigger set. The function β seems to remain a metric in this case also.

Suppose we have some metric D between formulas. D-distance alone does not give detailed comparison of the behavior of formulas with respect to its evaluation for different assignments. However, applied to residues this measure permits to described some kind of context of formula evaluations. For a given formula Φ and for a given $k \leq |\boldsymbol{Var}(\Phi)|$, denote by $\mathcal{A}(\Phi, k)$ all its maximal sat-assignments of length $\leq k$. The metric space $\mathcal{M}_k(\Phi) =_{df} (\{\Phi(A) : A \in \mathcal{A}(\Phi, k)\}, D)$ represents, in some way, the general structure of the evaluation of Φ for assignments from $\mathcal{A}(\Phi, k)$. If for a relatively small ε these spaces have a subexponential ε-net then this could give new ideas for constructing algorithms or proofs.

For a set $\mathcal{S}$ of formulas with n variables the spaces $\mathcal{M}_k(\Phi)$, $\Phi \in \mathcal{S}$, represent a global structure of assignment for these formulas. These spaces may be converted into one metric space with Hausdorff or Gromov-Hausdorff metric [3], 7.3.1–7.3.2. And after that we can continue a more global metric analysis of the structure of the problem.

5. CONCLUDING REMARKS

The notions introduced above are far from being finalized. An important experimental work in needed, that is analysis of concrete algorithms and problems.

The similarity measures introduced above are based on bijective embedding. It may happen that similarity measure based on approximate homomorphisms are more adequate. It is particulary probable for the analysis of SAT.

Similarity measures based on approximate homomorphisms may be not metrical, however, this does not exclude that productive measures of information, based on such an approach, are possible.

REFERENCES

[1] A. N. Kolmogorov and V. M. Tikhomirov. ε-entropy and ε-capacity of sets in function spaces. *Uspekhi Mat. Nauk*, 14(2(86)):3–86, 1959. In Russian. English translation in: Selected Works of A.N. Kolmogorov: Volume III: Information Theory and the Theory of Algorithms (Mathematics and its Applications), Kluwer Academic Publishers, 1992.

[2] N. F. G. Martin and J. W. England. *Mathematical Theory of Entropy*. Addison-Wesley, 1981.

[3] D. Burago, Yu. Burago, and S. Ivanov. *A Course in Metric Geometry*, volume 33 of *Graduate Studies in Mathematics*. Americal Mathematical Society, 2001.