Instance Complexity of NP-hard sets

Martin Mundhenk *

May 5, 1999

Abstract

Instance complexity was introduced by Orponen, Ko, Schöning, and Watanabe [7, 14, 15] as a measure of the complexity of individual instances of a decision problem. Comparing instance complexity to Kolmogorov complexity (i.e. the inherent complexity of a string), they introduced the notion of p-hard instances, and conjectured that every set not in P has p-hard instances. Whereas this conjecture is still unsettled, Fortnow and Kummer [4] proved that NP-hard sets have p-hard instances, unless P=NP. The unbounded version of the conjecture was proven wrong by Kummer [9], and therefore it cannot be used to characterize e.g. the class recursive sets.

In [13] a slightly weaker notion of hard instances was introduced. In the unbounded version, it is used to characterize the classes of recursively enumerable resp. recursive sets. In time-bounded versions, the class P could be characterized- Hard instances are shown to be stronger than complexity cores (introduced by Lynch [12]). Nevertheless, NP-hard sets must have both: super-polynomially dense hard instances and super-polynomially dense non-hard instances, unless P=NP.

This paper is a survey of results on Instance Complexity, especially focusing on NP-hard sets.

*Universität Trier, FB IV – Informatik, D-54286 Trier, Germany, `mundhenk@ti.uni-trier.de`.

1 Introduction

There are two perhaps complementary aspects to understand the computational intractability of decision problems. The "traditional" aspect considers the problem collectively. It considers a single program which decides all instances and asks how well the program fares as it gets larger and larger instances. This is the customary mode of thought in complexity theory, where the asymptotic worst-case behaviour of programs is emphasized. The tractability of a problem is seen to depend on the distribution of the positive and negative instances. Because feasible programs can determine only easy distributions, intractable problems underly more complex distributions. Under this aspect, hardness is a property of the problem as a whole, and the question why a problem is hard asks for distributional properties of the problem. But this does not always reflect our experience with hard problems. For example, consider the set SAT of satisfiable Boolean formulae. Beside its hardness, there are many tractable instances, e.g. Horn formulae. Hence, the collective hardness of SAT does not depend on this certain instances, because it does not decrease if we remove all of it from SAT. This gave rise to another aspect which considers the instances individually. There, hardness is a property of individual instances, and the question why a problem is hard asks for hard instances.

A first approach to study instances which make a problem intractable has been via the notion of *polynomial complexity cores* introduced by Lynch [12]. A polynomial complexity core for a problem A is a set of instances C such that every program for A needs superpolynomial time for all but finitely many instances in C. Lynch showed that every intractable decidable problem has an infinite complexity core. Intuitively, Horn formulae are not in a core C_{SAT} for SAT, because we can construct a program for SAT which runs in polynomial time on all Horn formulae. But the "all but finitely many" provision in the definition allows any finite variation of C_{SAT} to be a core for SAT. Hence, for every Horn formula ϕ, the set $C_{\text{SAT}} \cup \{\phi\}$ is again a core for SAT, which contradicts our above intuition. On the other hand, for every $\psi \in C_{\text{SAT}}$ we can construct a program for SAT which decides ψ in linear time by table look up. Therefore, the "all but finitely many" provision cannot be dropped from the definition. Concluding, every individual problem instance belongs to some core, and therefore the notion of a complexity core does not seem to lead to any useful formulations of the idea of hard instances.

The possibility of patching algorithms by finite tables is the main difficulty in formulating a satisfactory notion of hard instances. However the difficulty can be overcome by taking also the sizes of programs into account. This led to the notion of instance complexity [15], which uses ideas stemming from the study of time-bounded Kolmogorov complexity (introduced by Hartmanis [5]).

2 Kolmogorov Complexity

Kolmogorov complexity measures the information content of strings. Originally, it was introduced independently by Kolmogorov [8] and Chaitin [3] to give precise computational meaning to the concept of randomness and information contained in a finite string. An excellent source for it is [11]. Let us consider a simple example. The string 00000000000000000000 is easily seen to be described by "20 0s", whereas the string 10010110100111100010 has no clear pattern which allows to give such a short description. Therefore, the last string contains "more information" than the first one. To give a numerical value to the information content of a string, Kolmogorov complexity defines it as the length of the shortest program which prints the string. For example, the string s consisting only of 0s is printed by the program "`for i:=1 to` n`do print "0" od`", where n is the length of s. This program has length $28 + \log n$. Another program that prints s is "`print` s" which has length $6 + n$. This shows that the Kolmogorov complexity of a string has about the length of the string as upper bound. If one chooses any universal programming language, it does up to an constant summand not depend on that choice (see [5, 15, 11] for details).

We leave out details like the choice of a universal Turing machine with respect to which the size and the computation time of the programs are measured, and suppose that an "optimal interpreter" is used to run the programs. The existence, robustness, and invariance of such an interpreter and more details can be found e.g. in [5, 15, 11]. For simplicity and w.l.o.g. we assume that the program $\pi = \varepsilon$ denoted by the empty string halts on every input without any computation and output.

The *Kolmogorov complexity* $\mathrm{C}(x)$ of a string $x \in \Sigma^*$ is the length of the shortest program which gets no input and prints x, i.e.

$$\mathrm{C}(x) = \min_{\text{program } \pi} \{|\pi| \mid \pi \text{ outputs } x\} \quad .$$

We also consider relativized Kolmogorov complexity,

$$\mathrm{C}^A(x) = \min_{\text{program } \pi} \{|\pi| \mid \pi \text{ with oracle } A \text{ outputs } x\} \quad .$$

The notion of time-bounded Kolmogorov complexity was introduced by Hartmanis [5]. The *t-time bounded Kolmogorov complexity* is is the length of the shortest program which prints x in time $t(|x|)$, i.e.

$$\mathrm{C}^t(x) = \min_{\text{program } \pi} \{|\pi| \mid \pi(\varepsilon) = x \text{ and } \pi(\varepsilon) \text{ makes at most } t(|x|) \text{ steps}\} \quad .$$

We use $\mathrm{C}^{A,t}$ to denote the time-bounded Kolmogorov complexity relative to oracle A.

As mentioned above, the Kolmogorov complexity of a string has about the length of the string as an upper bound,

$$\exists c\ \forall x\ \forall A :\ \mathrm{C}^{A,t}(x) \leq |x| + c$$

for time bounds $t(n) \geq n$. This shows that the time-bounded Kolmogorov complexity is computable: given $|x|$, run all programs up to length $|x| + c$ for time $t(|x|)$ and look for the shortest program with output x. The (unbounded) Kolmogorov complexity is not computable.

Using counting arguments, one obtains certain lower bounds. There are fewer programs (=strings) of length $< n$ than strings of length n. Hence, there exist strings which reach this upper bound,

$$\forall n\ \exists x \in \Sigma^n :\ \mathrm{C}^{A,t}(x) \geq |x| \quad .$$

For each length, there is only a finite number of programs of that length. Therefore, the Kolmogorov complexity has no limit inferior,

$$\forall n\ \exists x_0\ \forall x \geq x_0 :\ \mathrm{C}^{A,t}(x) \geq n \quad .$$

Moreover, the function

$$\mu_C(x) \ = \ \min\{\mathrm{C}(y) \mid y \geq x\}$$

is not computable, has no limit, and grows slower than any computable unlimited function.

3 Instance complexity

Instance complexity [15] measures the individual decision complexity of strings with respect to a decision problem A. The measure used is the length of the shortest program which partially decides A and decides x. A program π which "partially decides" a problem A is allowed to accept or reject inputs, but also to remain undecided; this means, that it may answer "don't know" or run infinitely on certain inputs. On all inputs, which the program decides, the decision must be correct with respect to A. Such a program π is called *A-consistent*, denoted $\pi \simeq A$. The instance complexity of x with respect to A is denoted $\mathrm{ic}(x : A)$,

$$\mathrm{ic}(x : A) = \min_{\text{program } \pi} \{|\pi| \mid \pi \simeq A \wedge \pi(x) = A(x)\} \quad .$$

The Kolmogorov complexity of a string is its upper bound for the instance complexity with respect to any set A. This is, because a program which, on any input, computes a string x and either accepts (resp. rejects) if the input equals x, or answers "don't know" if the input not equals x, is an A-consistent program for every A containing x (resp. for every A not containing x). Therefore,

$$\exists c\ \forall A\ \forall x :\ \mathrm{ic}(x : A) \leq \mathrm{C}(x) + c \quad .$$

Similar as for Kolmogorov complexity, a time-bounded version of instance complexity is used. The *t-time bounded instance complexity of a string x w.r.t. A* is

$$\mathrm{ic}^t(x : A) = \min_{\text{program } \pi} \{|\pi| \mid \quad \pi \simeq A\ \wedge\ \pi(x) = A(x)\ \wedge \\ \pi \text{ on input } x \text{ halts after at most } t(|x|) \text{ steps}\}$$

Similar as above, time-bounded instance complexity is bounded by time-bounded Kolmogorov complexity. Because the constructed program deciding x simulates a t-time bounded computation of x, in [15] for the instance complexity a slightly larger time bound t' is obtained (where e.g. $t'(n) = ct(n)\log t(n) + c$),

$$\exists c\ \forall A\ \forall x :\ \mathrm{ic}^{t'}(x : A) \leq \mathrm{C}^t(x) + c \quad .$$

Since it is undecidable, whether a program is A-consistent, the time-bounded instance complexity is uncomputable. However, it can be approximated for recursive A. From a program π which is A-consistent on all instances of length n, we can construct a program π' which behaves like π on all instances of length n and answers "don't know" on all instances of other lengths. This program π' has length about $|\pi| + \log n$. Hence, for every recursive A there exists a computable function f such that for every recursive A there exists a ϕ (which is interpreted as program for A) such that forall x

$$\mathrm{ic}^t(x : A) \;\leq\; f(x, \phi) \;\leq\; \mathrm{ic}^t(x : A) + \log |x| \quad .$$

4 The Instance Complexity Conjecture

The complexity class P of problems decidable in polynomial time consists of exactly those problems which have polynomially time-bounded instance complexity bounded by a constant. This can be seen by the following arguments. If problem A is in P, then A is decided by some polynomially time-bounded program π, and its size $|\pi|$ bounds the instance compleity of all instances with respect to A. The other way round, up to a given length k, there is only a finite set of programs of length at most k. If all instances of a problem A have instance complexity at most k, there is a finite collection of A-consistent programs, and every instance x is decided by at least one of these programs. Therefore the new program, which on input x runs all of these programs one after the other until a program decided x, decided every instance correctly with respect to A.

Proposition 4.1 *[15] A problem A is in* P *if and only if there exists a polynomial p and a constant k such that for all instances x,* $\mathrm{ic}^p(x : A) \leq k$.

Hence, for a problem out of P, for every polynomial p and constant k there exist x with $\mathrm{ic}^p(x : A) > k$. This is a tempting way to define hard instances, but it is misleading. Eventually, it yields the same difficulties like with complexity cores.

Proposition 4.2 *[15] A set C is a polynomial complexity core for A if and only if for every polynomial p, for every constant k, and for almost every $x \in C$,* $\mathrm{ic}^p(x : A) > k$.

The difficulty lies in the constant k, which is somehow out of control. Because basically $\mathrm{ic}^p(x : A) \leq |x|$, it is clear that $\mathrm{ic}^p(x : A) \leq k$ for almost every k. To obtain a reasonable notion of hard instances, it is therefore necessary to replace k by something dependent on x. Moreover, it is conjectured that strings with Kolmogorov complexity close to the maximum (so called *Kolmogorov random strings* x with $\mathrm{C}(x) \geq |x|$) essentially contribute to the hardness of problems, because these strings contain information which cannot be used by feasible algorithms (note the parallel "distributional" believe that intractable sets encorporate distributions of elements which are not feasibly recognizable). This led to a notion of polynomially hard instances

Definition 4.3 *[15] A problem A has p-hard instances, if for every polynomial p there exists a polynomial p' and a constant c such that for infinitely many x,* $\mathrm{ic}^p(x : A) \geq \mathrm{C}^{p'}(x) - c$.

and to the *Instance Complexity Conjecture*

Conjecture 4.4 *[15] Every problem A not in* P *has p-hard instances.*

5 Problems with unbounded hard instance

Proposition 4.1 can be translated directly to a characterization of the class of recursive problems: a problem A is recursive if and only if for a constant k and for all x, $\mathrm{ic}(x : A) \leq k$. Kummer [9] proved a kind of "Gap Theorem". For every k there are only finitely many instances x with instance complexity between k and $C(x)$.

Theorem 5.1 *[9] A problem A is recursive if and only if*
$\mathrm{ic}(x : A) \leq \log \mathrm{C}(x) - 1$ *for almost all x.*

Allowing only a slightly larger instance complexity, we already obtain nonrecursive sets.

Theorem 5.2 *[9] There is a nonrecursive, recursively enumerable problem A and a constant c such that* $\mathrm{ic}(x : A) \leq \log \mathrm{C}(x) + c$ *for all x.*

This proves that the respective Instance Complexity Conjecture for recursive problems – a problem A is recursive iff $\mathrm{ic}(x : A) \geq C(x)$ for infinitely many x – does not hold. But it turned out that it holds for problems being complete for the class of recursively enumerable sets. Buhrman and Orponen [2] considered the instance complexity of the set R of Kolmogorov random strings, $R = \{x \mid \mathrm{C}(x) \geq |x|\}$. The complement of the set R is known to be complete for the class of recursively enumerable sets under Turing reductions. They showed that "random strings make hard instances," i.e. for a constant c, $\mathrm{ic}(x : R) \geq \mathrm{C}(x) - c$ for every $x \in R$. The same conclusion could be drawn for many-one complete problems.

Theorem 5.3 *[2, 9] For every recursively enumerable many-one complete problem A there is a constant c such that* $\mathrm{ic}(x : A) \geq \mathrm{C}(x) - c$ *for infinitely many x.*

This proves the respective Instance Complexity Conjecture for recursively enumerable complete problems.

Theorems 5.2 and 5.3 support the intuition that hard problems distinguish between instances with high and with low Kolmogorov complexity, whereas less hard problems cannot do so (cf. the notion of distinguishing complexity in [10]). In order to obtain characterizations of classes in terms of comparisons of instance and Kolmogorov complexity, it therefore seems reasonable to replace the measure of *absolute* Kolmogorov complexity by a *relative* one. In the following characterization of recursive sets, the Kolmogorov complexity relative to the Halting Problem K_0 and to the set under consideration is taken.

Theorem 5.4 *[13] A problem A is recursive if and only if for almost every x,* $\mathrm{ic}(x : A) \leq \mathrm{C}^{K_0 \oplus A}(x)$.

(Formally, $K_0 \oplus A = \{0w \mid w \in K_0\} \cup \{1w \mid w \in A\}$. Intuitively, the oracle $K_0 \oplus A$ answers queries to oracle K_0 and questions to oracle A.)

We sketch the proof, because the technique is typical for many proofs concerning the construction of instances with complexity than its Kolmogorov complexity (e.g. [6, 15, 9, 4, 13]). As argued above, recursive sets have instance complexity bounded by a constant. Because $\mathrm{C}^{K_0 \oplus A}(x)$ has no limit inferior, the given upper bound for recursive sets holds. Assume that A is not recursive. We show that program N (see Figure 1) with the halting problem K_0 and A as oracles outputs hard instances.

On input 0^n, it removes inductively all programs of length $\leq n$ which are not A-consistent, until it finds an instance of length $\geq n$, which is not

```
input 0^n
Π := Σ^{≤n}                  (* set of programs to be checked for A-consistency *)
for m := n, n+1, n+2, ... do
  for all π ∈ Π and all y ∈ Σ^m do
    if π(y) ≄ A(y) then Π := Π − {π} end
  end
  if ∃y, |y| = m ∀π ∈ Π :  π(y) =⊥ then
    output the lexicographically first such y ∈ A^{=m} and halt
  end
end
```

Figure 1: Program N

---

decided by any of the remaining programs. Such an instance exists, because otherwise a finite collection of programs would have been found, which is A-consistent and decides all instances. Therefore, N halts on every input 0^n. The output has instance complexity $> n$, because no A-consistent program of length $\leq n$ decides it. The Kolmogorov complexity (relative to $K_0 \oplus A$) of the output is bounded by the Kolmogorov complexity of 0^n, which is about $\mathrm{C}(0^n) \leq \log n$, plus the length of the program N, which is constant. Hence, the instance complexity of almost all outputs is greater than its relativized Kolmogorov complexity. $\square$

Interestingly, we get a characterization of the recursively enumerable sets, if the bound on the instance complexity is applied on the elements of the set only.

Theorem 5.5 *[13] A problem A is recursively enumerable if and only if for almost every $x \in A$:* $\mathrm{ic}(x : A) \leq \mathrm{C}^{K_0 \oplus A}(x)$.

6 Problems with p-hard instances

We now turn to polynomially time-bounded instance complexity, and to the Instance Complexity Conjecture. The class E is the class of problems decidable deterministically in time $2^{O(n)}$, and NE is its non-deterministic variant. Because $\mathrm{P} \neq \mathrm{E}$, problems complete for E are not in P. These problems have p-hard instances.

Theorem 6.1 *[15] Every* E*-complete problem has p-hard instances.*

Moreover, in [2] it is shown that E-complete problems have many p-hard instances, namely at least $2^{\varepsilon n}$ many of length n, for an $\varepsilon > 0$ dependent on the set. Hence, for E-complete problems a strong version of the Instance Complexity Conjecture holds.

Other than in the unbounded case, problems A with logarithmically small polynomially time-bounded instance complexity are not known to be in P. These problems are said to have *low instance complexity.* But at least they cannot be NP complete.

Theorem 6.2 *[15] For no* NP *complete problem A and polynomial p holds* $\mathrm{ic}^p(x : A) = O(\log|x|)$ *for almost every x, unless* $\mathrm{P} = \mathrm{NP}$.

Moreover, problems with low instance complexity are relatively close to problems in P (see [1]). Note that for every polynomial p, $\log \mathrm{C}(x) \leq \log \mathrm{C}^p(x) \leq \log|x|$. Therefore, problems with low instance complexity cover the polynomially time-bounded cases of both Theorems 5.1 and 5.2. It is still open, whether $\mathrm{ic}^p(x : A) \leq \log \mathrm{C}^{p'}(x)$ (for polynomials p and p') implies that A is in P.

Similar as in the unbounded case, the Instance Complexity Conjecture holds for problems, which are NP complete in a natural way.

Theorem 6.3 *[4] Every recursive problem $A \notin$* P *which is* NP *hard with respect to honest polynomially time-bounded Turing reductions, has p-hard instances.*

The proof of this theorem uses a program to find hard instances which has a similar structure as the algorithm given in Figure 1. In fact, the polynomial time bounds, which must be respected, make it much more involved. The oracle A can be dropped, because only recursive problems are considered. And the oracle K_0, which could generally have been replaced by an NP complete problem in the polynomially time-bounded case, can be dropped, because one can take advantage of a self-reducibility structure of the NP hard problem A. This information is used to find the instances which are not decided by the collection of programs kept. Therefore, if the set of candidates for undecided instances can be calculated in polynomial time, there is no need that the set A under consideration is NP hard. This is the case for tally sets $A \subseteq 0^*$ (i.e. sets which are defined over a one-element alphabet instead of an (at least) two-element alphabet).

Theorem 6.4 *[4] Every recursive tally problem $A \notin \mathrm{P}$ has p-hard instances.*

As a consequence of Theorem 6.3 we obtain that the Instance Complexity Conjecture holds if $\mathrm{P} = \mathrm{NP}$. Essentially, much better results cannot be obtained with known techniques, because the conjecture fails in a relativized world [4]. Therefore, it is interesting to consider weaker formulations of the conjecture. Dropping the polynomial time-bound for the Kolmogorov complexity yields

Theorem 6.5 *[13] A recursive problem A is in P if and only if for a polynomial p and for almost every x, $\mathrm{ic}^p(x : A) \leq \mathrm{C}(x)$.*

This yields the following notion of hard instances.

Definition 6.6 *The set of* hard instances *for a recursive problem A is*

$$H(A) := \{x \mid \text{ for every polynomial } q : \mathrm{ic}^q(x : A) > \mathrm{C}(x)\}.$$

The Theorem 6.5 says that every recursive set not in P has hard instances. We now compare hard instances to polynomial complexity cores. By Proposition 4.2, hard instances are a polynomial complexity core. But the set of hard instances can be much smaller than a core. In [13] it is shown that there exist problems whith sparse hard instances but Σ^* as polynomial complexity core. Nevertheless, similar as for polynomial complexity cores, NP-hard problems cannot have sparse hard instances unless $\mathrm{P} = \mathrm{NP}$.

References

[1] V. Arvind, J. Köbler, and M. Mundhenk. Upper bounds for the complexity of sparse and tally descriptions. *Mathematical Systems Theory*, 29:63–94, 1996.

[2] H. Buhrman and P. Orponen. Random strings make hard instances. In *Proc. 9th Structure in Complexity Theory Conference*, pages 217–222. IEEE, 1994.

[3] G.J. Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM*, 13:547–569, 1966.

[4] L. Fortnow and M. Kummer. On resource-bounded instance complexity. *Theoretical Computer Science*, 161:123–140, 1996.

[5] J. Hartmanis. Generalized Kolmogorov complexity and the structure of feasible computations. In *Proc. 24th IEEE Symp. on Foundations of Computer Science*, pages 439–445, 1983.

[6] K. Ko. A note on the instance complexity of pseudorandom sets. In *Proc. 7th Structure in Complexity Theory Conference*, pages 327–337. IEEE, 1992.

[7] K. Ko, P. Orponen, U. Schöning, and O. Watanabe. What is a hard instance of a computational problem? In *Proc. Structure in Complexity Theory*, pages 197–217. Springer Verlag, 1986.

[8] A.N. Kolmogorov. Three approaches for defining the concept of information quantity. *Problems Information Transmission*, 1:1–7, 1965.

[9] M. Kummer. Kolmogorov complexity and instance complexity of recursively enumerable sets. *SIAM Journal on Computing*, 25(6):1123–1143, 1996.

[10] S. Laplante and J. Rogers. Indistinguishability. Technical Report 96-26, University of Chicago Dept. of Computer Science, 1996.

[11] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Springer-Verlag, 1997.

[12] N. Lynch. On reducibility to complex or sparse sets. *Journal of the ACM*, 22:341–345, 1975.

[13] M. Mundhenk. On hard instances. *Theoretical Computer Science*, 1999.

[14] P. Orponen. On the instance complexity of NP-hard problems. In *Proc. 5th Structure in Complexity Theory*, pages 20–27. IEEE, 1990.

[15] P. Orponen, K. Ko, U. Schöning, and O. Watanabe. Instance complexity. *Journal of the ACM*, 41:96–121, 1994.