WHICH ALGORITHMS ARE FEASIBLE? MAXENT APPROACH

D.E. COOKE, V. KREINOVICH, AND L. LONGPRÉ
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA†

Abstract. It is well known that not all algorithms are feasible; whether an algorithm is feasible or not depends on how many computational steps this algorithm requires. The problem with the existing definitions of feasibility is that they are rather *ad hoc*. Our goal is to use the maximum entropy (MaxEnt) approach and get more motivated definitions.

If an algorithm is feasible, then, intuitively, we would expect the following to be true: *If we have a flow of problems with finite average length $\bar{l}$, then we expect the average time $\bar{t}$ to be finite as well.*

Thus, we can say that an algorithm is *necessarily feasible* if $\bar{t}$ is finite for *every* probability distribution for which $\bar{l}$ is finite, and *possibly feasible* if $\bar{t}$ is finite for *some* probability distribution for which $\bar{l}$ is finite.

If we consider *all* possible probability distributions, then these definitions trivialize: every algorithm is possibly feasible, and only linear-time algorithms are necessarily feasible.

To make the definitions less trivial, we will use the main idea of MaxEnt and consider only distributions for which the entropy is the largest possible. Since we are interested in the distributions for which the average length is finite, it is reasonable to define MaxEnt distributions as follows: we fix a number l_0 and consider distributions for which the entropy is the largest among all distributions with the average length $\bar{l} = l_0$.

If, in the above definitions, we only allow such "MaxEnt" distributions, then the above feasibility notions become non-trivial: an algorithm is possibly feasible if it takes exponential time (to be more precise, if and only if its average running time $\bar{t}(n)$ over all inputs of length n grows slower than some exponential function C^n), and necessarily feasible if it is *sub-exponential* (i.e., if $\bar{t}(n)$ grows slower than *any* exponential function).

†Emails: {dcooke,vladik,longpre}@cs.utep.edu

Key words: maximum entropy, feasible algorithm, average computational complexity, Moore's law

1. When is an algorithm feasible? The problem, and what we plan to do

1.1. SOME ALGORITHMS ARE NOT FEASIBLE

It is well known that not all algorithms are feasible (see, e.g., [3,9,10]): whether an algorithm is feasible or not depends on how many computational steps this algorithm requires.

Let us give two examples.

1.2. EXAMPLE OF A FEASIBLE ALGORITHM

First, we will give an example of a feasible algorithm. Let us consider the following simple sorting algorithm for ordering a finite list of n numbers (this algorithm is called *selection sort*):

- First, we find the largest number. To do this, we check the elements one by one and after each check, select the largest-of-checked number. Initially, when we start the sorting, the first element is the largest-of-checked. If we have already checked the first k numbers, and we know which of them is the largest, all we have to do to find out which number is the largest among the first $k+1$ is compare the existing "champion" with the newly checked number:
  - If the champion is larger, it stays the champion.
  - If the new number is larger, it becomes a new champion.

  After we have checked all n numbers, the largest-of-checked becomes simply the largest of all of them. So, we can place this number at the right end of the desired list, and place the number, that was originally at this right end, into the champion's original location.
- After the above-described first iteration, the largest element is in its desired place. Now, we apply a similar algorithm to the remaining $n-1$ elements, find the largest of them, and place it right before the largest one.
- Then, we repeat the same procedure with $n-2$ elements, etc.

How many computational steps does this algorithm require? On each iteration, we add exactly one element to the list of already sorted ones, so, we need at most n iterations. On each of these iterations, we check at most n elements, so we need at most n computational steps per iteration. Thus, this algorithm requires a total computation time of $\leq n \times n = n^2$. This algorithm is quite realistic and thus, feasible.

In practice, for large lists, better sorting algorithms are used, with computation time $n \cdot \log(n)$; they are, clearly, also quite feasible.

1.3. EXAMPLE OF A NON-FEASIBLE ALGORITHM

Let us now give an example of a non-feasible algorithms.

There exists an algorithm for solving all problems from elementary geometry [18]. It is known that this algorithm requires, in some cases, at least doubly exponential time [2], i.e., time of order 2^{2^n}. Even for small n, this amount of time is much larger than the lifetime of the Universe divided by the smallest physically possible time interval (of $\approx 10^{-43}$ seconds). Thus, this algorithm is clearly non-feasible.

1.4. IMPORTANT COMMENT: "NON-FEASIBLE" DOES NOT NECESSARILY MEAN "USELESS"

It is important to remark that the very fact that an algorithm requires a long time for *some* examples does not necessarily make it practically unusable: it may be that these examples are rare, and in almost all practical cases, this algorithm can be successfully applied.

1.5. THE EXISTING DEFINITION OF FEASIBILITY, ITS ADVANTAGES AND DRAWBACKS

In modern theory of computation, an algorithm is usually called *feasible* if is is *polynomial-time*, i.e., if its running time is bounded by some polynomial of the length of the input. This definition has its pluses and minuses:

- This definition has many *advantages*, e.g., it is *independent* on the type of the computational device that we are using.
- On the other hand, this definition is not completely satisfactory, mainly because we are trying to formalize one of the major theoretical concepts of computing theory, but the existing definition of feasibility is rather *ad hoc*.

In the following text, we will use the maximum entropy (MaxEnt) approach [5–8] to get more motivated definitions.

2. Our main idea

If an algorithm is feasible, then, intuitively, we would expect the following to be true:

> *If we have a flow of problems with finite average length, then we expect the average time to be finite as well.*

3. Formalization: general idea

How can we formalize this statement?

- Let $l(x)$ denote the length of the word x,
- let $t_U(x)$ denote the running time of the algorithm U on the input x, and
- let $p(x)$ be the probability (frequency) of the word x in the flow.

In these notations:

- the average word length is $\bar{l} = \sum p(x) \cdot l(x)$, and
- the average time is $\bar{t} = \sum p(x) \cdot t_U(x)$.

Thus, a natural formalization of the above principle takes the following form:

- We somehow select a family $\mathcal{P}$ of *natural* probability distributions $\{p(x)\}$.
- We require that for every distribution p from the family $\mathcal{P}$ for which the average word length $\bar{l}$ is finite, the average computation time $\bar{t}$ is also finite.

4. Formalization: first approximation

The problem with this definition is that we do not know which probability distributions are natural. There are two extreme approaches to this problem:

- the *most cautious* approach is to consider all possible probability distributions, and to require that $\bar{t}$ is finite for *all* probability distributions for which $\bar{l}$ is finite;
- the *least cautious* approach is to require that $\bar{t}$ is finite for *some* probability distribution for which $\bar{l}$ is finite.

The corresponding two definitions lead to the following results:

Definition 1. *We say that an algorithm is necessarily feasible if $\bar{t}$ is finite for every probability distribution for which $\bar{l}$ is finite.*

Definition 2. *We say that an algorithm is possibly feasible if $\bar{t}$ is finite for some probability distribution for which $\bar{l}$ is finite.*

Proposition 1.

- *An algorithm U is necessarily feasible if and only if it is linear-time, i.e., if there exists a constant C for which $t_U(x) \le C \cdot l(x)$ for all inputs x.*
- *Every algorithm U is possibly feasible.*

Comment. For the readers' convenience, all the proofs are given at the end of the paper.

4.1. THE PROBLEMS WITH THIS DEFINITION

Proposition 1 is not very satisfactory:

- Linear time seems too restrictive; we would like the definition to be more restrictive.
- On the other hand, *no* restrictions seems too non-restrictive. We would like *some* restrictions.

5. MaxEnt approach

5.1. MAIN IDEA

To make the definitions less trivial, we will use the main idea of MaxEnt: instead of all possible probability distributions, we will only consider *MaxEnt distributions*,

i.e., distributions for which the entropy $S = -\sum p(x) \cdot \log(p(x))$ is the largest possible.

5.2. MAIN DEFINITIONS

Since we are interested in the distributions for which the average length is finite, it is reasonable to define MaxEnt distributions as follows:

Definition 3.

- *Let l_0 be a real number. By a MaxEnt distribution corresponding to l_0 we mean the distribution for which the entropy is the largest among all distributions with the average length $\bar{l} = l_0$.*
- *By a MaxEnt distribution, we mean a distribution that is MaxEnt for some real number $l_0 > 0$.*

Definition 4. *We say that an algorithm is necessarily MaxEnt-feasible if $\bar{t}$ is finite for every MaxEnt distribution for which $\bar{l}$ is finite.*

Definition 5. *We say that an algorithm is possibly MaxEnt-feasible if $\bar{t}$ is finite for some MaxEnt distribution for which $\bar{l}$ is finite.*

5.3. AUXILIARY DEFINITIONS

To describe what algorithms are feasible in the sense of these distributions, we will need the following auxiliary definition:

Definition 6. *By an average running time of an algorithm U, we mean the function $\bar{t}_U(n)$ defined as*

$$\bar{t}_U(n) = \frac{1}{N_n} \cdot \sum_{x:l(x)=n} t_U(x),$$

where the sum is taken over all words x of length $l(x) = n$, and N_n is the total number of such words.

Now, we are ready to formulate the main result:

5.4. MAIN RESULT

Proposition 2.

- *An algorithm U is necessarily MaxEnt-feasible if and only if it is average subexponential-time, i.e., if for all $C > 0$,*

$$\lim_{n\to\infty} \frac{\bar{t}_U(n)}{\exp(C \cdot n)} = 0.$$

- *An algorithm U is possibly MaxEnt-feasible if and only if it is average exponential-time, i.e., if there exist constants C_0 and C_1 for which $\bar{t}_U(n) \le C_0 \cdot \exp(C_1 \cdot n)$ for all positive integers n.*

6. Discussion and future plans

6.1. NICE PROPERTIES OF THE ABOVE DEFINITIONS AND RESULTS

In contrast to traditional treatment of feasibility, we gain a formal distinction between the two levels of *possible* and *necessary* feasibility, the discussion intuited, e.g., by M. Gell-Mann [4], but never before properly formalized (see, e.g., [1]).

6.2. POSSIBLE PROBLEMS (?) WITH THESE DEFINITIONS AND RESULTS

Our result seemingly contradicts the traditional definition of feasible as polynomial time.

However, there is no contradiction if we take into consideration the so-called *Moore's law*, according to which the computer speed grows exponentially with time [11,12,16]. In view of this law, linear *time* means actually exponentially many computational steps, and therefore, a problem that requires exponentially many computational steps can be actually solved in reasonable time [17].

(Also, on a more speculative note, some exponential-time algorithms become quite feasible if we take the curvature of space-time into consideration [13].)

6.3. POSSIBILITIES OF MAKING THESE DEFINITIONS MORE ADEQUATE

Unless we invoke Moore's law, the above formalization is not completely adequate: e.g., an algorithm that require 2^n steps on inputs of length n is treated as possibly feasible. This may be not restrictive enough, because, such an algorithm, for an input of a reasonable length $n \approx 300$, would need more computational steps than there can be during the lifetime of our Universe.

Even sub-exponential does not necessarily mean practically feasible, because, e.g., a function $2^{\sqrt{n}}$ is sub-exponential, but grows pretty fast with n.

If we want to make the above definitions more restrictive (or at least more realistic), it is desirable to find and formalize other intuitive statements about feasibility (examples of such statements are given, e.g., in [14,15]). The fact that our intuitive statement led to reasonable restriction shows that MaxEnt is indeed a reasonable technique to use.

Acknowledgments

This work was supported in part by NSF under grants No. EEC-9322370 and No. DUE-9750858, by NASA under cooperative agreement NCCW-0089, and by Future Aerospace Science and Technology Program (FAST) Center for Structural Integrity of Aerospace Systems, effort sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number F49620-95-1-0518.

The authors are thankful to all the participants of MaxEnt'97 workshop, especially to Anton Garrett and John Skilling, for valuable discussions.

References

1. D. E. Cooke, *Scientific Research: From the Particular to the General*, El Paso Energy Award for Research Excellence Presentation, University of Texas at El Paso, April 9, 1997.
2. J. H. Davenport and J. Heintz, "Real quantifier elimination is doubly exponential", *Journal of Symbolic Computations*, 1988, Vol. 5, No. 1/2, pp. 29–35.
3. M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*, W. F. Freeman, San Francisco, 1979.
4. M. Gell-Mann, *The Quark and the Jaguar: Adventures in the Simple and the Complex*, Freeman, N.Y., 1994.
5. K. M. Hanson and R. N. Silver (Eds.), *Maximum Entropy and Bayesian Methods*, Kluwer Academic Publishers, Dordrecht, 1996.
6. E. T. Jaynes, "Information theory and statistical mechanics", *Phys. Rev.*, 1957, Vol. 108, pp. 171–190.
7. E. T. Jaynes, "Where do we stand on maximum entropy?", In: R. D. Levine and M. Tribus (Eds.) *The maximum entropy formalism*, MIT Press, Cambridge, MA, 1979.
8. V. Kreinovich, "Maximum entropy and interval computations", *Reliable Computing*, 1996, Vol. 2, No. 1, pp. 63–79.
9. H. R. Lewis and C. H. Papadimitriou, *Elements of the Theory of Computation*, Prentice-Hall, Inc., New Jersey, 1981.
10. J. C. Martin, *Introduction to languages and the theory of computation*, McGraw-Hill, N.Y., 1991.
11. G. E. Moore, "Cramming more components onto integrated circuits", *Electronics Magazine*, 1965, Vol. 38, No. 8, pp. 114–117.
12. G. E. Moore, "Lithography and the Future of Moore's Law", In *Proceedings of the SPIE Conference on Optical/Laser Microlithography, February 1995*, SPIE Publ., Vol. 2440, 1995, pp. 2–17.
13. D. Morgenstein and V. Kreinovich, "Which algorithms are feasible and which are not depends on the geometry of space-time", *Geombinatorics*, 1995, Vol. 4, No. 3, pp. 80–97.
14. H. T. Nguyen and V. Kreinovich, "When is an algorithm feasible? Soft computing approach", *Proceedings of the Joint 4th IEEE Conference on Fuzzy Systems and 2nd IFES*, Yokohama, Japan, March 20–24, 1995, Vol. IV, pp. 2109–2112.
15. H. T. Nguyen and V. Kreinovich, "Towards theoretical foundations of soft computing applications", *International Journal on Uncertainty, Fuzziness, and Knowledge-Based Systems*, 1995, Vol. 3, No. 3, pp. 341–373.
16. R. R. Schaller, "Moore's law: past, present, and future", *IEEE Spectrum*, June 1997, pp. 53–59; see also discussion on p. 8 of the August 1997 issue of *IEEE Spectrum*.
17. D. Schirmer and V. Kreinovich, "Towards a More Realistic Definition of Feasibility", *Bulletin of the European Association for Theoretical Computer Science (EATCS)*, 1996, Vol. 90, pp. 151–153.
18. A. Tarski, *A decision method for elementary algebra and geometry*, 2nd ed., Berkeley and Los Angeles, 1951.

Appendix: Proofs

6.4. PROOF OF PROPOSITION 1

For a linear time algorithm, $t_U(x) \le C \cdot l(x)$ and therefore, for every probability distribution, $\bar{t} \le C \cdot \bar{l}$. So, if $\bar{l}$ is finite, the average time $\bar{t}$ is also finite.

Let us show that if an algorithm is not linear-time, then there exists a probability distribution for which $\bar{l}$ is finite, but $\bar{t}$ is infinite. Indeed, linear-time means that there exists a constant C for which, for all x, $t_U(x)/l(x) \le C$. Therefore, the fact that an algorithm is not linear-time means that for every possible constant C, in particular, for $C = 2^n$, there exists a word x_n for which $t_U(x_n)/l(x_n) > 2^n$. Let

us take the probability distribution that is concentrated on each x_n with a probability $p_n = N \cdot 2^{-n}/l(x_n)$, where N is a normalization constant (which guarantees that $\sum p_n = 1$). Since $\sum 2^{-n} = 1 < \infty$, and $l(x_n) \geq 1$, we have

$$\sum \frac{2^{-n}}{l(x_n)} \leq \sum 2^{-n} < \infty$$

and therefore, such a normalization is indeed possible.

For this probability distribution,

$$\bar{l} = \sum p_n \cdot l(x_n) = \sum N \cdot 2^{-n} = N \cdot \sum 2^{-n} < \infty.$$

However, since $t_U(x_n)/l(x_n) \geq 2^n$, we get

$$\bar{t} = \sum p_n \cdot t_U(x_n) = \sum N \cdot 2^{-n} \cdot \frac{t_U(x_n)}{l(x_n)} \geq N \cdot \sum 1 = \infty.$$

The first part of Proposition 1 is proven.

To show that every algorithm is possibly feasible in the sense of Definition 2, it is sufficient to take a probability distribution that is concentrated on one word x_0 with probability 1. Then, $\bar{l} = l(x_0) < \infty$ and $\bar{t} = t_U(x_0) < \infty$. The proposition is proven.

6.5. PROOF OF PROPOSITION 2

Every MaxEnt probability distribution is a solution to the constrained optimization problem $S = -\sum p(x) \cdot \log(p(x)) \to \max$ under the constraints $\sum p(x) = 1$ and $\sum l(x) \cdot p(x) = l_0$. Lagrange multiplier method reduces this formula to the *unconstrained* optimization problem

$$-\sum [p(x) \cdot \log(p(x)) + \lambda \cdot p(x) + \mu \cdot p(x) \cdot l(x)] \to \max.$$

If we differentiate the resulting objective function with respect to $p(x)$ and equate the result to 0, we get $-\ln(p(x)) + 1 + \lambda + \mu \cdot l(x) = 0$, i.e., $p(x) = N \cdot \exp(B \cdot l(x))$, where B is a real number and N is the normalization constant (that guarantees $\sum p(x) = 1$).

What are the possible values of B? The value $p(x)$ depends only on the length $n = l(x)$ of the word x: $p(x) = N \cdot \exp(B \cdot n)$. For every length $l(x) = n$, there are exactly $N_n = A^n$ words of length n, where A is the number of symbols in the input alphabet. Thus, the sum p_n of the probabilities $p(x)$ for all words x of length n is equal to $p_n = N \cdot A^n \cdot \exp(B \cdot n) = N \cdot \exp(\ln(A) \cdot n) \cdot \exp(B \cdot n) = N \cdot \exp(n \cdot (B + \ln(A))$. The total probability is equal to the sum of these terms: $1 = \sum p_n$. Thus, for the sum of these terms p_n to converge, we must have $B + \ln(A) < 0$. It is easy to see that for an arbitrary positive real number $C > 0$, if we take $B + \ln(A) = -C$ (i.e., $B = -C - \ln(A)$), then we get a MaxEnt distribution.

For an arbitrary MaxEnt distribution, the average time $\bar{t} = \sum p(x) \cdot t_U(x)$, which is defined over all possible words, can also be represented as the sum of the following sub-sums: $s = s_1 + s_2 + \ldots + s_n + \ldots$, where:

- the sum s_1 is taken over all the words of length 1,
- the sum s_2 is taken over all the words of length 2,
- ...,
- the sum s_n is taken over all the words of length n,
- ...

For a fixed n, the values $p(x) = N \cdot \exp(B \cdot n)$ in the sum $s_n = \sum p(x) \cdot t_U(x)$ do not depend on x, so, $s_n = p(x) \cdot \sum t_U(x) = N \cdot \exp(B \cdot n) \cdot sumt_U(n)$. The sum of running times is, in its turn, equal to $N_n \cdot \bar{t}_U(n) = A^n \cdot \bar{t}_U(n)$. Thus, $s_n = N \cdot A^n \cdot \exp(B \cdot n) \cdot \bar{t}_U(n) = N \cdot \exp(-C \cdot n) \cdot \bar{t}_U(n)$. Thus, the convergence of the series $s_1 + \ldots + s_n + \ldots$ means that the sum $\sum \exp(-C \cdot n) \cdot \bar{t}_U(n)$ converges.

Let us show that this conclusion proves Proposition 2:

- If an algorithm U is possibly MaxEnt-feasible, then this series converge for some $C > 0$. From the convergence of the series, we can conclude that the term tends to 0, i.e., that $\bar{t}_U(n)/\exp(C \cdot n) \to 0$ as $n \to \infty$. Since this sequence tends to 0, there exists a constant C_0 that bounds it. Hence, $\bar{t}_U(n) \le C_0 \cdot \exp(C \cdot n)$, i.e., U is indeed an average exponential-time algorithm.
- Vice versa, let U be an average exponential-time algorithm, i.e., let $\bar{t}_U(n) \le C_0 \cdot \exp(C_1 \cdot n)$ for all n. Then, if we take $C = C_1 + \varepsilon$ for some $\varepsilon > 0$, we conclude that $\bar{t}_U(n) \cdot \exp(-C \cdot n) \le C_0 \cdot \exp(-\varepsilon \cdot n)$ converges, and thus, this algorithm is feasible with respect to the corresponding MaxEnt distribution.
- If an algorithm is necessarily MaxEnt-feasible, then for *every* $C > 0$, we get $\bar{t}_U(n)/\exp(C \cdot n) \to 0$ as $n \to \infty$, and thus, the algorithm U is indeed average subexponential-time.
- Vice versa, let us show that every subexponential-time algorithm is necessarily MaxEnt-feasible, i.e., that the sum $\sum \exp(-C \cdot n) \cdot \bar{t}_U(n)$ converges for all $C > 0$. Indeed, since U is an average subexponential-time algorithm, we have $\bar{t}_U(n)/\exp((C/2) \cdot n) \to 0$ and therefore,

$$\frac{\bar{t}_U(n)}{\exp((C/2) \cdot n)} \le C_0$$

for some $C_0 > 0$ and for all n. Thus, $\bar{t}_U(n) \le C_0 \cdot \exp((C/2) \cdot n)$ and $s_n = \bar{t}_U(n) \cdot \exp(-C \cdot n) \le C_0 \cdot \exp(-(C/2) \cdot n)$. The terms s_n are bounded by a convergent sum and thus, the sum $\sum s_n$ also converges.

The proposition is proven.