A New Graph Characteristic And Its Application to Numerical Computability

Frank Harary[1], Vladik Kreinovich[2], and Luc Longpré[2]

[1]Department of Computer Science
New Mexico State University
Las Cruces, NM 88003, USA
fnh@nmsu.edu

[2]Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968, USA
{vladik,longpre}@cs.utep.edu

Abstract

Many traditional numerical algorithms include a step on which we check whether a given real number a is equal to 0. This checking is easy for rational numbers, but for constructive *real* numbers, whether a number is 0 or not is an algorithmically undecidable problem. It is therefore desirable to re-formulate the existing algorithms with as few such comparisons as possible. We describe a new graph characteristic; this characteristic describes how the number of comparisons in an algorithm can be reduced.

1 A Numerical Computation Problem

1.1 Comparison with 0

This is an important part of numerical algorithms, but in general, algorithmically undecidable.

Many traditional numerical algorithms include a step in which we check whether or not a certain real number is equal to 0. Even a standard algorithm for solving a linear equation $ax + b = 0$ requires that we check whether $a = 0$ and whether $b = 0$. If $a \neq 0$, then this equation has a single solution $x = -b/a$; if $a = 0$ and $b = 0$, then every real number x is a solution; and if $a = 0$ and $b \neq 0$, then this equation has no solutions at all.

For *rational* numbers a, it is algorithmically possible to check whether $a = 0$ or not. However, not all real numbers are rational. One can define a *constructive* (*computable*) real number a as a real number for which there exists an algorithm generating, for every integer k, a rational number a_k for which $|a_k - a| \leq 2^{-k}$. (This number a_k is called a 2^{-k}-*rational approximation* to a.) It is known that no general algorithm can tell whether a given constructive real number a is equal to 0. See [1, 2, 3, 4, 8] and references therein. Strictly speaking, this means that no guaranteed (100% reliable) algorithm can solve an arbitrary linear equation with constructive coefficients.

1.2 We wish to minimize the number of such comparisons

Since solving linear equations is, in practice, easy, from the practical viewpoint, this theoretical impossibility is not as serious as other negative theoretical results which reflect true practical impossibility. To emphasize the fact that comparing a real number with 0 is not such a hard task, computer scientists have analyzed the notion of "conditional" computations, i.e., those computations which require, at some step, comparing some real numbers to 0. In other words, these computations require a hypothetical device which, given a constructive real number a, checks whether or not this real number is equal to 0. In the language of theoretical computer science, this hypothetical device is called an *oracle*, and conditional computations are computations with respect to this oracle. Most classical numerical algorithms which use such a comparison can be described as such conditional computations.

Traditionally, analysis of this problem has been about whether a given conditional algorithm can be reformulated without such an oracle. As a result of this analysis, it turned out that for several important numerical problems, the use of this oracle is unavoidable. Since the comparison problem is algorithmically undecidable, each use of this oracle (e.g., each comparison of a real number with 0) can become a computational stumbling block; therefore, it is desirable to have as few such comparisons in our algorithm as possible. Such a minimization is described, in detail, in [5], in a more general algorithmic context than validated numerical methods.

1.3 A nontrivial example of such minimization

Let us start with a simple result about such a minimization, which, in numerical terms, looks as follows. We will illustrate with three numbers. Suppose that in a numerical algorithm, we must first check whether each of three given numbers x_1, x_2, and x_3 is equal to 0. It turns out that by using bisection, we can reduce these three checks to two. To be more precise, we use bisection to determine how many of the three numbers x_i are equal to 0. In principle, there are four possibilities: none, one, two, or all three of the numbers x_i can be equal to 0. So, to apply bisection, we must check whether at least two of these numbers are equal to, i.e., whether the following statement is true:

$$(x_1 = 0 \,\&\, x_2 = 0) \vee (x_1 = 0 \,\&\, x_3 = 0) \vee (x_2 = 0 \,\&\, x_3 = 0).$$

This statement can be reduced to a formula $z = 0$ if we take into consideration that $A = 0 \,\&\, B = 0$ is equivalent to $A^2 + B^2 = 0$, and $A = 0 \vee B = 0$ is equivalent to $A \cdot B = 0$. Thus, the above statement can be brought to a form $z = 0$, with

$$z = (x_1^2 + x_2^2) \cdot (x_1^2 + x_3^2) \cdot (x_2^2 + x_3^2).$$

After we check whether $z = 0$, it is sufficient to ask one more question to find the number of zero x_i's: if $z = 0$, then we have to ask whether all three are equal to 0; if $z \neq 0$, then we have to ask whether at least one is equal to 0. Thus, after two questions of the type $z = 0$, we get the number of zero x_i's. To find out which x_i are equal to 0, we can now start computing all three real numbers x_i with better and better accuracy by computing rational approximations x_{ik} to x_i. It is known that $a = 0$ if and only if for some k, the rational 2^{-k}-approximation a_k to a exceeds 2^{-k}, i.e., $|a_k| > 2^{-k}$. Therefore, if, e.g., two of the three numbers x_i are different from 0, then eventually, for two of these numbers, we will have $|x_{ik}| > 2^{-k}$. At this point, we stop computing x_{ik}, because we know which of the numbers x_i are equal to 0, and which are not.

Similarly, if we need to check whether each of $n = 2^k - 1$ numbers equal to 0, we can use bisection to reduce this problem to k checks of equality to 0. A much more technical result shows that in general, we cannot reduce $2^k - 1$ to less than k checks. In general, n independent checks can be reduced to $k = \lceil \log_2(n+1) \rceil$ dependent ones.

A similar reduction is possible for algorithms which solve differential equations. In these algorithms, we often need to check whether a constructive *function* is identically equal to 0 or not. Similarly, we can reduce the checking of $n = 2^k - 1$ functions to k checks, and once we know how many of given functions f_i are identical to 0, we can find exactly which functions are equal to 0 by computing, for different rational numbers r and different accuracies k, the 2^{-k}-approximations $f_{ik}(r)$ to the values $f_i(r)$. One can show that a function is not identical to 0 if and only if for some r and k, we have $|f_{ik}(r)| > 2^{-k}$.

1.4 Open problem

If we have n independent checks, i.e., checks in which each checked number is given from the very beginning and does not depend on the results of other checks, then we can replace them by $k = \lceil \log_2(n+1) \rceil$ dependent checks. In a general algorithm, some checked numbers may be given from the very beginning, and some other checked numbers may depend on the results of the previous checks (as in the above reduced computations, the second check depends on whether $z = 0$ or not). In such a general situation, how can we decrease the number of checks?

The general dependence between the checks can be described by a directed graph (*digraph*) $D = (V, E)$. We refer to books [6, 7] for digraph theory.

- The vertices V are numbers that, in the course of running the algorithm, we must compare with 0, and

- we place an arc $(a, b) = a \to b$ from a vertex a to a vertex b if the number checked at vertex b depends on the result of checking whether $a = 0$.

We next describe a possible reduction for a general digraph.

2 Description of the New Graph Characteristic and the Main Result

To formulate our result, we introduce the following notion:

Definition 1. *Let D be a digraph (V, E).*

- *If arc $(a, b) \in E$, we say that a is a parent of b, and b is a child of a.*
- *By a stratification of the digraph D, we mean a partition $V = V_1 \cup \ldots \cup V_n$ of the set of all vertices V into subsets V_i (called strata) such that every parent of every vertex $v \in V_i$ belongs to one of the sets V_j, $j < i$. (In particular, it means that vertices from V_1 do not have parents at all.)*

For each vertex v, we can define $d(v)$ as such i for which $v \in V_i$. For this function $d : V \to \{1, \ldots, n\}$, if $a \to b$, then $d(a) < d(b)$. Vice versa, for every mapping d with this property, we can define a stratification as $V_i = \{v \mid d(v) = i\}$. Thus, a stratification can be alternatively defined as a mapping d with the above property.

Definition 2. *Let D be a digraph.*

- *By the log-size $\ell(A)$ of a finite abstract set A with $N = |A|$ elements, we mean the number $\ell(A) = \lceil \log_2(N + 1) \rceil$.*
- *By the log-size $\ell(V_1, \ldots, V_n)$ of a stratification, we mean the sum $\ell(V_1) + \ldots + \ell(V_n)$ of log-sizes of its strata.*
- *By the log-size of a digraph, we mean the smallest possible log-size of its stratifications.*

If we divide a set into two pieces, then the sum of the logarithms of their sizes is equal to the logarithm of their product and is therefore larger than the logarithm of the size of the original set. So, to keep the log-size to a minimum, it is reasonable not to artificially subdivide each stratum, but to keep the strata as large as possible. At first glance, it may therefore seem that to obtain the minimum log-size, we should take as V_1 the largest possible first set, i.e., the set of all vertices with no parents; as V_2, all vertices whose only parents are in V_1, etc. However, this stratification does not always lead to the smallest log-size. For example, if we have a 2-layer digraph, with four vertices on the top layer L_1 and two vertices on the second layer L_2 (Fig. 1), then the above "natural" stratification $V_1 = L_1$ and $V_2 = L_2$ (Fig. 2) leads to log-size $\ell(V_1) + \ell(V_2) = \lceil \log_2(4 + 1) \rceil + \lceil \log_2(2 + 1) \rceil = 3 + 2 = 5$. If we move one of the vertices from L_1 into the second stratum V_2 (Fig. 3), we get $|V_1| = 4 - 1 = 3$, $|V_2| = 2 + 1 = 3$, and a smaller log-size $\ell(V_1) + \ell(V_2) = \lceil \log_2(3 + 1) \rceil + \lceil \log_2(3 + 1) \rceil = 2 + 2 = 4 < 5$.

$$\begin{array}{cccc} L_1 & L_1 & L_1 & L_1 \\ \searrow & \swarrow & \searrow & \swarrow \\ & L_2 & & L_2 \end{array} \qquad \begin{array}{cccc} V_1 & V_1 & V_1 & V_1 \\ \searrow & \swarrow & \searrow & \swarrow \\ & V_2 & & V_2 \end{array} \qquad \begin{array}{cccc} V_2 & V_1 & V_1 & V_1 \\ \searrow & \swarrow & \searrow & \swarrow \\ & V_2 & & V_2 \end{array}$$

Fig. 1 Fig. 2 Fig. 3

We discovered the notion of log-size based on our numerical computation problem, we cannot (yet) give an intuitive geometric interpretation of this characteristic. However, we *can* give a geometric *analogue* of this characteristic: namely, the log-size is somewhat similar to the logarithm of the number of maximal paths in a digraph: Indeed, if we have several layers with $n_1, n_2, \ldots, n_m$ vertices respectively, and each vertex in each layer is connected with each vertex in the next layer, then the total number of paths is equal to $n_1 \cdot \ldots \cdot n_m$, and hence it logarithm is equal to $\log_2(n_1) + \ldots + \log_2(n_m)$. This expression is clearly similar to the above expression $\lceil \log_2(n_1 + 1) \rceil + \ldots + \lceil \log_2(n_m + 1) \rceil$ (of course, although these expressions are similar, they are different).

Definition 3.

- *Let D be a digraph. By an elementary transformation, we mean a transformation of D into a new digraph D' in which:*
  - *in one arc $a \to b$, we replace b by two new vertices b_0 and b_1;*
  - *there is no arc from a to b_0 or b_1;*
  - *every arc from $c \neq a$ to b leads to two new arcs: from c to b_0 and from c to b_1, and*
  - *every arc from b to c leads to two new arcs: from b_0 to c and from b_1 to c.*

  In symbols, $D' = (V', E')$, where $V' = V - \{b\} \cup \{b_0, b_1\}$, and

$$E' = E - \{(c,b) \mid c \in V\} - \{(b,c) \mid c \in V\} \cup$$
$$\{(c,b_0) \mid (c,b) \in E \,\&\, c \neq a\} \cup \{(c,b_1) \mid (c,b) \in E \,\&\, c \neq a\} \cup$$
$$\{(b_0,c) \mid (b,c) \in E\} \cup \{(b_1,c) \mid (b,c) \in E\}$$

- *By a transformation of a digraph D, we mean either the digraph D itself, or a digraph obtained from D by a sequence of elementary transformations.*

- *By the NC-size of a digraph, we mean the smallest log-size of its transformations.*

Although an elementary transformation increases the number of vertices, it can decrease the log-size of a digraph. As an example, let us consider the 2-layer digraph from Fig. 4, with five vertices $v_{11}, \ldots, v_{15}$ on the top layer L_1 and eight vertices $v_{21}, \ldots, v_{28}$ on the second layer L_2.

$$v_{11} \quad v_{12} \quad v_{13} \quad v_{14} \quad v_{15}$$
$$\swarrow \searrow \quad \downarrow \quad \swarrow \searrow \quad \downarrow \quad \swarrow \searrow$$
$$v_{21} \; v_{22} \; v_{23} \; v_{24} \; v_{25} \; v_{26} \; v_{27} \; v_{28}$$

Fig. 4

For this digraph, none of the elements from L_2 can be included in V_1, and thus, as one can easily check, the log-size is obtained when $V_1 = L_1$ and $V_2 = L_2$, and is equal to $\ell(V_1) + \ell(V_2) = \lceil \log_2(5+1) \rceil + \lceil \log_2(8+1) \rceil = 3 + 4 = 7$. If we apply an elementary transformation to the arc $v_{13} \to v_{24}$, we get a new digraph D' with $5 + 2 = 7$ elements $v_{24,0}, v_{11}, \ldots, v_{15}, v_{24,1}$ on the top layer, and $8 - 1 = 7$ elements on the second layer: $v_{21}, v_{22}, v_{23}, v_{25}, \ldots, v_{28}$ (Fig. 5). For this new digraph D', the natural stratification into these two layers leads to a smaller log-size: $\ell(V_1) + \ell(V_2) = \lceil \log_2(7+1) \rceil + \lceil \log_2(7+1) \rceil = 3 + 3 = 6 < 7$.

$$v_{24,0} \quad v_{11} \quad v_{12} \quad v_{13} \quad v_{14} \quad v_{15} \quad v_{24,1}$$
$$\swarrow\searrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \swarrow\searrow$$
$$v_{21} \; v_{22} \; v_{23} \; v_{25} \; v_{26} \; v_{27} \; v_{28}$$

Fig. 5

The acronym NC comes from *N*umerical *C*omputations; the reason for this becomes clear from the following result:

Theorem 1. *Let D be a digraph describing dependency between zero-checking in a numerical algorithm. Then, this algorithm can be replaced by a new algorithm with the number of zero-checks equal to the NC-size of the digraph D.*

Proof. Let us first show that we can reduce the number of zero-checks to the log-size of the digraph.

Indeed, let $V_1, \ldots, V_n$ be a stratification which corresponds to this log-size, i.e., whose log-size is equal to the log-size of the graph. Let $|V_i| = n_i$ and $\sum n_i = n$. Then, by definition of a stratification, the values from V_1 do not depend on anything at all, so we can use the above bisection and replace checking all numbers from V_1 by checking $\lceil \log_2(n_1 + 1) \rceil = \ell(V_1)$ numbers.

By the same definition of a stratification, the numbers from the stratum V_2 depend only on V_1. Thus, after we have checked all the numbers from V_1, we know the values to check in V_2. Hence, we have n_2 known numbers to check, and we can use the above bisection to replace the checking of all numbers from V_2 by checking $\lceil \log_2(n_2+1) \rceil = \ell(V_2)$ numbers. Similarly, after we have checked V_2, we can check all the numbers from V_3, etc. Totally, we need $\ell(V_1) + \ldots + \ell(V_n)$ checks, and this is exactly the log-size of the digraph.

To complete the proof, let us now show that we can also restrict the number of checks by using a digraph obtained via an elementary transformation. Indeed, let $a \to b$ be an arc, i.e., let the number checked at b depend on the result of checking whether $a = 0$. In principle, we do not have to wait until we checked $a = 0$ to check the number on b-stage: instead, we can consider both possibilities $a = 0$ and $a \neq 0$, generate two numbers b_1 and b_0 that will be checked correspondingly when $a = 0$ and when $a \neq 0$, and check both. If b depends on some other c, then this dependence still stands for both b_0 and b_1. The theorem is proven.

Conclusion

We have presented a new characteristic of a digraph, and we have shown that this characteristic is useful for numerical computations. It is therefore important to analyze this characteristic, hopefully find its more intuitive form, and learn how to compute it rapidly.

Acknowledgments. This work was supported in part by NASA under cooperative agreement NCC5-209, by NSF grant No. DUE-9750858 and CDA-9522207, by the United Space Alliance grant No. NAS 9-20000 (PWO C0C67713A6), by the 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, by the National Security Agency under Grant No. MDA904-98-1-0564, by National Security Agency under Grant No. MDA904-98-1-0564, and by the Hong Kong grant RGC4138/97E and UGC Direct Grant 2050148. The authors are thankful to Bill Gasarch (University of Maryland) for valuable discussions.

Part of this work was conducted while one of the authors (V.K.) was visiting the Department of Mechanical and Automation Engineering at the Chinese University of Hong Kong under the support of grant RGC4138/97E.

References

[1] M. J. Beeson, *Foundations of Constructive Mathematics*, Springer-Verlag, New York, 1985.

[2] E. Bishop, *Foundations of Constructive Analysis*, McGraw-Hill, New York, 1967.

[3] E. Bishop, D. S. Bridges, *Constructive Analysis*, Springer-Verlag, New York, 1985.

[4] D. S. Bridges, *Constructive Functional Analysis*, Pitman, London, 1979.

[5] W. I. Gasarch and G. A. Martin, *Bounded Queries in Recursion Theory*, Birkhäuser, Boston, 1998.

[6] F. Harary, *Graph Theory*, Addison-Wesley, Reading, MA, 1969.

[7] F. Harary, R. Z. Norman, and D. Cartwright, *An Introduction to the Theory of Directed Graphs*, Wiley, New York, 1965.

[8] V. Kreinovich, A. Lakeyev, J. Rohn, and P. Kahl, *Computational Complexity and Feasibility of Data Processing and Interval Computations*, Kluwer, Dordrecht, 1997.