Supporting Information

Papadimitriou 10.1073/pnas.1416954111

SI Text

Complexity Definitions

An algorithm is an unambiguous sequence of elementary computational steps that, when applied on an input (a given string of symbols), always eventually terminates with an output (another string of symbols), which stands in a particular relationship to the input; this relationship characterizes the algorithm (and in practice is its *raison d' être*). If y is the output of algorithm A on input x, we sometimes write $y = A(x)$. There are many standard formal mathematical models, such as the Turing machine (main text), which capture the notion of an algorithm, and all (with the fascinating exception of the quantum computation model, which is conjectured to be strictly more powerful than the rest when it comes to polynomial time computation) agree on the essential points.

Algorithms are evaluated with respect to the time they require. We say that algorithm A operates in time $f(n)$, where $f(n)$ is a function mapping the set of nonnegative integers to itself, if when A is presented with an input of size n (a string with n symbols, that is), it always operates in time $f(n)$. An algorithm that operates in in time $p(n)$, where p is some fixed polynomial such as n^3, is called a polynomial-time algorithm.

A problem—what an algorithm solves—is a particular specification of how the output of the algorithm must stand in relation to its input. In particular, a search problem Π_C is defined in terms of a polynomial time algorithm C, which, given as input two strings x, y, such that the length of y is at most a polynomial function of the length of x, produces one of the two outputs: "yes" or "no." Π_C is then the following problem: Given an input x, produce an output y such that algorithm C, on input x, y, outputs yes— such an output is called a witness of x with respect to problem Π_C. Or, if no witness y of x exists, produce the output "no witness exists."

The class of all search problems is denoted **NP**; the 10 problems identified in the main text are well-known examples of search problems. Now **P** is the class of all search problems that can be solved by polynomial time algorithms; again, a few examples are given in the main text. (Traditionally, the classes **NP** and **P** are defined not as classes of search problems, but as classes of languages—that is to say, infinite sets of strings of symbols. The version of the formalism presented here focuses on output-producing algorithms and is more appropriate for the focus of this paper.)

A reduction from search problem Π to search problem Ψ is a pair of polynomial time algorithms R and T, with the following property: Given an input x of problem Π, $R(x)$ is an input of problem Ψ; and if y is a witness of $R(x)$ with respect to problem Ψ, then $T(y)$ is a witness of x with respect to the original problem Π. Intuitively, a reduction from Π to Ψ is a method for establishing that Ψ is "no easier" than Π. A problem Ψ is called **NP**-complete if it has the following property: For any search problem Π, there is a reduction from Π to Ψ. Thus, an **NP**-complete problem is, informally, at least as hard as any problem in **NP**; it is included among the members of **NP** that are least likely to be in **P**.

We are mainly interested in the behavior of algorithms for larger and larger inputs, but occasionally it is useful to define algorithms with inputs and outputs in binary and of fixed input and output size. Such algorithms are called circuits, because they can be rendered as combinational Boolean circuits with the following kinds of gates: n input gates, "and" gates, "or" gates, and "not" gates. n of these gates are also designated output gates. Each gate feeds into other, noninput gates, in such a way that no cycles are created. Not gates have only one other gate feeding into them.

Circuits are used in the definition of the classes **PPAD** and **PLS**. The problem "End of the line" is the following: For some n, we are given a directed graph on 2^n nodes, implicitly defined through two circuits with n inputs and outputs, called S and P (for successor and predecessor). If $u \neq v$ are elements of $\{0,1\}^n$, we say that the graph contains the edge (u, v) if $S(u) = v$—that is, the circuit S with inputs set to u has output v—and $P(v) = u$. Obviously, this graph has at most one arrow entering, and at most one arrow leaving, each node. Circuit P is such that $S(0^n) = 0^n$, and so the all-zeros node has no predecessor — it is a source. The problem is this: Given S and P, find another source besides 0^n or find a sink (a vertex with no outgoing edges). It is easy to see that one of the two must exist.

Definition 1: **PPAD** is the class of all search problems that can be reduced to End of the line.

The class **PLS** is defined in an analogous way. We are given two circuits, S (for successor) and F (for potential function). The directed graph again has 2^n vertices, identified with all strings with symbols 0 and 1 of length n, and has an arrow from u to v if the following two conditions hold: $S(u) = v$ and $F(v) > F(u)$, where $u > v$ means that the two integers coded in binary by u and v are so related. Sink is the following problem: Given two circuits S and F, find a sink of the corresponding graph. It is again easy to see that a sink must exist.

Definition 2: **PLS** is the class of all search problems that can be reduced to Sink.

Fitness Functions That Are Additive on Pairs of Loci

For a population with m genes with two alleles each, a fitness function is any function f mapping $\{0,1\}^m$ to the nonnegative reals. Our proposed family of fitness functions capturing pairwise locus interactions is defined as follows: Consider a weighted undirected graph (undirected lines called edges connecting vertices), where the genes are the vertices, and the set of edges is denoted by E. Assume that all genes have two alleles, denoted by 0 and 1. For each edge $e = \{i, j\}$ (recall that an edge is a set of two genes) and each pair of $0-1$ values for x_i and x_j, there is a fixed a nonnegative weight $w^e_{x_i,x_j}$. Then for all $x_1, \ldots, x_m \in \{0,1\}$, the fitness or the genotype $(x_1, \ldots, x_m)$ is defined as

$$f(x_1, \ldots, x_m) = \sum_{\{ij\}=e\in E} w^e_{x_i,x_j}.$$

The proof of *Theorem 8* now proceeds as follows: We start from the problem "Locally maximum weighted cut": Given a graph with weights on its edges, find a partition of the vertices into two sets such that migrating any single vertex from one set to the other does not increase the total weight of edges joining vertices in different sets. This problem is known to be **PLS** complete. Furthermore, this problem can be reduced to the problem of finding a pure genotype that is a stable point of Eq. **5** of the main text through the following transformation: Given a graph with m nodes and weights c_e on the edges, create a fitness function with pairwise interactions between m genes, based on the same graph, and with weights $w^e_{x_i,x_j} = 1 + s \cdot c_e$ if $x_i \neq x_j$ and $w^e_{x_i,x_j} = 1$ otherwise, where the selection strength s is small, say m^{-2}. It is easy to see that a pure stable point of Eq. **5** corresponds to a locally maximum max cut of the original weighted graph.