# Combinatorial Property Testing (a survey)*

Oded Goldreich†
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, ISRAEL.
E-mail: oded@wisdom.weizmann.ac.il.

May 5, 1998

**Abstract**

We consider the question of determining whether a given object has a predetermined property or is "far" from any object having the property. Specifically, objects are modeled by functions, and distance between functions is measured as the fraction of the domain on which the functions differ. We consider (randomized) algorithms which may query the function at arguments of their choice, and seek algorithms which query the function at relatively few places.

We focus on combinatorial properties, and specifically on graph properties. The two standard representations of graphs – by adjacency matrices and by incidence lists – yield two different models for testing graph properties. In the first model, most appropriate for dense graphs, distance between $N$-vertex graphs is measured as the fraction of edges on which the graphs disagree over $N^2$. In the second model, most appropriate for bounded-degree graphs, distance between $N$-vertex $d$-degree graphs is measured as the fraction of edges on which the graphs disagree over $dN$.

To illustrate the two models, we survey results regarding the complexity of testing whether a graph is Bipartite. For a constant distance parameter, a constant number of queries suffice in the first model, whereas $\widetilde{\Theta}(\sqrt{N})$ queries are necessary and sufficient in the second model.

*Based mainly on joint work with Shafi Goldwasser and Dana Ron [17], and joint works with Dana Ron [19, 20].
†Currently visiting LCS, MIT. Supported by DARPA grant DABT63-96-C-0018.

# 1 Introduction and Summary of Known Results

The following general formulation of Property Testing was suggested in [17]:

> Let P be a fixed property of functions, and $f$ be an unknown function. The goal is to determine (possibly probabilistically) if $f$ has property P or if it is far from any function which has property P, where distance between functions is measured with respect to some distribution D on the domain of $f$. Towards this end, one is given examples of the form $(x, f(x))$, where $x$ is distributed according to D. One may also be allowed to query $f$ on instances of one's choice.

The above formulation is inspired by the PAC learning model [37]. In fact, property testing is related to variants of PAC learning as has been shown in [17]. The general formulation above allows the consideration of arbitrary distributions rather than uniform ones, and of testers which utilize only randomly chosen instances (rather than being able to query instances of their own choice). However, we do not consider these latter generalizations here, but rather focus on the special case (formulated previously in [34, 33]) where the distribution is uniform on the domain of the function, and testers are allowed to query the function on instances of their choice. Thus, the above formulation simplifies to the following definition, in which we associate a property with the class of functions satisfying it.

**Definition 1** (property tester)*: Let $S$ be a finite set, and* P *a subset of functions mapping $S$ to $\{0,1\}^*$. A* (property) tester for P *is a probabilistic oracle machine*[1]*, $M$, which given a* distance parameter $\epsilon > 0$ *and oracle access to an arbitrary function $f : S \mapsto \{0,1\}^*$ satisfies the following two conditions:*

1. (the tester accepts $f$ if it is in P)
   *If* $f \in \mathrm{P}$ *then* $\mathbf{Pr}(M^f(\epsilon)=1) \geq \frac{2}{3}$.
2. (the tester rejects $f$ if it is far from P)
   *If* $|\{x \in S : f(x) \neq g(x)\}| > \epsilon \cdot |S|$, *for every* $g \in \mathrm{P}$, *then* $\mathbf{Pr}(M^f(\epsilon)=1) \leq \frac{1}{3}$.

Property testing (as just defined) emerges naturally in the context of program checking [10, 29, 16, 34] and probabilistically checkable proofs (PCP) [6, 7, 14, 4, 3, 8, 11, 5, 9, 22, 24]. Specifically, in the context of program checking, one may choose to test that the program satisfies certain properties before checking that it computes a specified function. This paradigm has been followed both in the theory of program checking [10, 34], and in practice where often programmers first test their programs by verifying that the programs satisfy properties that are known to be satisfied by the function they compute. In the context of probabilistically checkable proofs, the property tested is being a codeword with respect to a specific code. This paradigm, explicitly introduced in [7], has shifted from testing codes defined by low-degree polynomials [6, 7, 14, 4, 3] to testing Hadamard codes [3, 8, 11, 5, 28, 36], and recently to testing the "long code" [9, 22, 24, 36].

Much of the work cited above deals with the development and analysis of testers for algebraic properties; specifically, linearity, multi-linearity, and low-degree polynomials [10, 29, 6, 7, 14, 16, 34, 4, 3, 8, 11, 5]. In contrast, following [17] we focus on testing combinatorial properties, and specifically on testing graph properties such as Bipartiteness.

[1] Alternatively, one may consider a RAM model of computation, in which trivial manipulation of domain and range elements (e.g., reading/writing an element and comparing elements) is performed at unit cost.

**The relevant parameters.** The main parameter relevant to property testing is the permitted *distance parameter*, denoted $\epsilon$. In addition, one may consider a confidence (or error bound) parameter, denoted $\delta$, rather than fixing an error bound of $\frac{1}{3}$ (as done in Definition 1 above). Standard amplification techniques do apply here, and thus we choose to fix the error bound. The complexity measures we focus on are the *query complexity* and the *running time* of the tester. All testers discussed in this survey have running-time which is substantially smaller than the full description of the function.

## 1.1 Motivation

The definition of property testing is a relaxation of the standard definition of a decision task: The tester is allowed arbitrary behavior when the object does not have the property, and yet is "close" to an object having the property. Thus, a property tester may be far more efficient than a standard decision procedure (for the same property).

In case the object is huge, as in case one thinks of a function and algorithms which operate in time polynomial in the length of the arguments to the function, there is actually no other alternative to property testing. That is, it is typically infeasible (i.e., requires exponential time in the length of the arguments) to decide whether such a function has the desired property. A property testing algorithm which operates in time polynomial in the length of the arguments thus offers a feasible approximation to a problem which is intractable in the exact formulation.

Property testers are valuable also in case one deals with objects of feasible size (i.e., size for which scanning the entire object is feasible): If a property tester is much faster than the exact decision procedure then it makes sense to run it before running the decision procedure. In case the object is far from having the property, we may obtain an indication towards this fact, and save the time we might have used running the decision procedure. In case the tester supplies proofs of violation of the property (as in some of the testers discussed below), we have obtain an absolutely correct answer without running the decision procedure at all. Thus, we may only need to run the decision procedure on objects which are close to having the property. In some natural setting where *typical* objects are either good (i.e., have the property) or very bad (i.e., are very far from objects having the property), we may gain a lot. Furthermore, *if* it is *guaranteed* that objects are either good or very bad then we may not even need to run the decision procedure at all. The gain in such a setting is enormous.

Being close to an object which has the property is a notion of approximation which, in certain applications, may be of great value. In some cases, being close to an object having the property translates to a standard notion of approximation. In other cases, it translates to a notion of "dual approximation". This point is clarified and exemplified below (by referring to specific properties). In both cases, a fast property tester which is more efficient than the decision procedure is of value, both if the decision procedure is feasible and more so if it is not.

Alternatively, we may be forced to take action, without having time to run a decision procedure, while given the option of modifying the object in the future, at a cost proportional to the number of added/omitted edges. For example, suppose you are given a graph which represents some design problem, where Bipartite graphs corresponds to a good design and changes in the design correspond to edge additions/omissions. Using a Bipartiteness tester you always accept a good design, and reject with high probability designs which will cost a lot to modify. You may still accept bad designs, but then you know that it will not cost you much to modify them later.

## 1.2 Testing Graph Properties – An Overview

As stated above, this survey focuses on testing graph properties. Two natural representations of a graph are offered by its adjacency matrix and by its incidence list. Correspondingly, we consider two representations of graphs by functions.

1. An $N$-vertex graph, $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, can be represented by the *adjacency predicate*, $f : \mathrm{V} \times \mathrm{V} \mapsto \{0, 1\}$, so that $(u, v) \in \mathrm{E}$ if and only if $f(u, v) = 1$.

2. An $N$-vertex graph of degree bound $d$, $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, can be represented by the *incidence function*, $g : \mathrm{V} \times [d] \mapsto \mathrm{V} \cup \{0\}$, so that $g(u, i) = v$ if $v$ is the $i^{\text{th}}$ vertex incident at $u$, and $g(u, i) = 0 \notin \mathrm{V}$ if $u$ has less than $i$ neighbors.

As usual, the choice of representation has a fundamental impact on the potential algorithm. Here the impact is even more dramatic since we seek algorithms which only inspect a relatively small fraction of the object (graph represented by a function). Furthermore, there is another fundamental impact of the choice of representation on the task of property testing. This has to do with our definition of distance, which is relative to the size of the domain of the function. In particular, distance $\epsilon$ in the first representation means a symmetric difference of $2\epsilon \cdot N^2$ edges, whereas in the second representation this means a symmetric difference of $2\epsilon \cdot dN$ edges. (In both cases, the extra factor 2 is due to the redundant representation which is adopted for sake of simplicity.)

As usual, the first representation (i.e., adjacency predicate) is most appropriate for dense graphs (i.e., $|\mathrm{E}| = \Omega(|\mathrm{V}|^2)$), whereas the second representation (i.e., incidence function) is applicable and most appropriate for graphs of degree bound $d$. We demonstrate the difference between the two representations by considering the task of testing whether a graph is Bipartite.

### 1.2.1 Some known results in the first (i.e., adjacency predicate) representation

Testers of complexity which depends only on the distance parameter, $\epsilon$, are known for several natural graph properties [17]. In particular, the following properties can be tested in query-complexity $\mathrm{poly}(1/\epsilon)$ and time complexity $\exp(\mathrm{poly}(1/\epsilon))$:

- $k$-Colorability, for any fixed $k \geq 2$. The query-complexity is $\mathrm{poly}(k/\epsilon)$, and for $k = 2$ the running-time is $\widetilde{O}(1/\epsilon^3)$. The Bipartite Tester is presented in Section 2.

- $\rho$-Clique, for any $\rho > 0$. That is, does the $N$-vertex graph have a clique of size $\rho N$.

- $\rho$-CUT, for any $\rho > 0$. That is, does the $N$-vertex graph have a cut of size at least $\rho N^2$. A generalization to $k$-way cuts works within query-complexity $\mathrm{poly}((\log k)/\epsilon)$.

- $\rho$-Bisection, for any $\rho > 0$. That is, can the vertices of the $N$-vertex graph be partitioned into two equal parts with at most $\rho N^2$ edges going between them.

**Remarks:**

1. For all the above properties, in case the graph has the desired property, the testing algorithm outputs some auxiliary information which allows to construct, in $\mathrm{poly}(1/\epsilon) \cdot N$-time, a partition which approximately obeys the property. For example, for $\rho$-CUT, we can construct a partition with at least $(\rho - \epsilon)N^2$ crossing edges.

2. Except for Bipartite testing, running-time of $\mathrm{poly}(1/\epsilon)$ is unlikely, as it will imply $\mathcal{NP} \subseteq \mathcal{BPP}$.

3. The $k$-Colorability tester has one-sided error: It always accepts $k$-colorable graphs. Furthermore, when rejecting a graph, this tester always supplies a $\mathrm{poly}(1/\epsilon)$-size subgraph which is not $k$-colorable. All other algorithms have two-sided error, and this is unavoidable within $o(N)$ query-complexity.

All the above property testing problems are special cases of the *General Graph Partition Testing Problem*, parameterized by a set of lower and upper bounds. In this problem one needs to determine whether there exists a $k$-partition of the vertices so that the number of vertices in each part as well as the number of edges between each pair of parts falls between the corresponding lower and upper bounds (in the set of parameters). A tester for the general problem has been presented in [17] too. The algorithm uses $\widetilde{O}(k^2/\epsilon)^{2k+O(1)}$ queries, runs in time exponential in its query-complexity, and has two-sided error.

Going beyond the General Graph Partition Problem, we remark that there are graph properties which are very easy to test in this model (e.g., Connectivity, Hamiltonicity, and Planarity) [17]. The reason being that for these properties either every $N$-vertex graph is at distance at most $O(1/N)$ from a graph having the desired property (and so for $\epsilon = \Omega(1/N)$ the trivial algorithm which always accepts will do), or the property holds only for sparse graphs (and so for $\epsilon = \Omega(1/N)$ one may reject any non-sparse graph). On the other hand, there are ("unnatural") graph properties in $\mathcal{NP}$ which are extremely hard to test; namely, any testing algorithm must inspect at least $\Omega(N^2)$ of the vertex pairs [17]. In view of the above, we believe that providing a characterization of graph properties, according to the complexity of testing them, may be very challenging.

**Relation to recognizing graph properties:** Our notion of testing a graph property P is a *relaxation* of the notion of *deciding the graph property* P which has received much attention in the last two decades [30]. In the classical problem there are no margins of error, and one is required to accept all graphs having property P and reject all graphs which lack it. In 1975 Rivest and Vuillemin [35] resolved the Aanderaa–Rosenberg Conjecture [32], showing that any deterministic procedure for deciding any non-trivial monotone $N$-vertex graph property must examine $\Omega(N^2)$ entries in the adjacency matrix representing the graph. The query complexity of randomized decision procedures was conjectured by Yao to be $\Omega(N^2)$. Progress towards this goal was made by Yao [38], King [27] and Hajnal [21] culminating in an $\Omega(N^{4/3})$ lower bound. This stands in striking contrast to the testing results of [17] mentioned above, by which some non-trivial monotone graph properties can be *tested* by examining a constant number of locations in the matrix.

**Application to the standard notion of approximation:** The relation of testing graph properties to the standard notions of approximation is best illustrated in the case of Max-CUT. Any tester for the class $\rho$-cut, working in time $T(\epsilon, N)$, yields an algorithm for approximating the maximum cut in an $N$-vertex graph, up to additive error $\epsilon N^2$, in time $\frac{1}{\epsilon} \cdot T(\epsilon, N)$. Thus, for any constant $\epsilon > 0$, using the above tester of [17], we can approximate the size of the max-cut to within $\epsilon N^2$ in constant time. This yields a constant time approximation scheme (i.e., to within any constant relative error) for dense graphs, improving over previous work of Arora *et. al.* [2] and de la Vega [13] who solved this problem in polynomial-time (i.e., in $O(N^{1/\epsilon^2})$–time and $\exp(\widetilde{O}(1/\epsilon^2)) \cdot N^2$–time, respectively). In the latter works the problem is solved by actually constructing approximate max-cuts. Finding an approximate max-cut does not seem to follow from the mere existence of a tester for $\rho$-cut; yet, as mentioned above, the tester in [17] can be used to find such a cut in time linear in $N$.

**Relation to "dual approximation"** (cf., [25, 26]): To illustrate this relation, we consider the $\rho$-Clique Tester mentioned above. The traditional notion of approximating Max–Clique corresponds to distinguishing the case in which the max-clique has size at least $\rho N$ from, say, the case in which the max-clique has size at most $\rho N/2$. On the other hand, when we talk of testing "$\rho$-Cliqueness", the task is to distinguish the case in which an $N$-vertex graph has a clique of size $\rho N$ from the case in which it is $\epsilon$-far from the class of $N$-vertex graphs having a clique of size $\rho N$. This is equivalent to the "dual approximation" task of distinguishing the case in which an $N$-vertex graph has a clique of size $\rho N$ from the case in which any $\rho N$ subset of the vertices misses at least $\epsilon N^2$ edges. To demonstrate that these two tasks are vastly different we mention that whereas the former task is NP-Hard, for $\rho < 1/4$ (see [9, 22, 23]), the latter task can be solved in $\exp(O(1/\epsilon^2))$-time, for any $\rho, \epsilon > 0$. We believe that there is no absolute sense in which one of these approximation tasks is more important than the other: Each of these tasks may be relevant in some applications and irrelevant in others.

### 1.2.2 Some known results in the second (i.e., incidence function) representation

Testers of complexity which depends only on the distance parameter, $\epsilon$, are known for several natural graph properties [19]. In particular, the following properties can be tested in time (and thus query-complexity) $\mathrm{poly}(d/\epsilon)$:

- *Connectivity.* The tester runs in time $\widetilde{O}(1/\epsilon)$. In case the graph is connected the algorithm always accepts, whereas in case the graph is $\epsilon$-far from being connected the algorithm rejects with probability at least $\frac{2}{3}$ and furthermore supplies a small counter-example to connectivity (in the form of an induced subgraph which is disconnected from the rest of the graph).

- *$k$-edge-connectivity.* The algorithms run in time $\widetilde{O}(k^3 \cdot \epsilon^{-3})$. For $k = 2, 3$ improved algorithms have running-times $\widetilde{O}(\epsilon^{-1})$ and $\widetilde{O}(\epsilon^{-2})$, respectively. Again, $k$-edge-connected graphs are always accepted, and rejection is accompanied by a counter-example.

- *$k$-vertex-connectivity, for $k = 2, 3$.* The algorithms run in time $\widetilde{O}(\epsilon^{-k})$.

- *Planarity.* The algorithm runs in time $\widetilde{O}(d^4 \cdot \epsilon^{-1})$.

- *Cycle-Freeness.* The algorithms run in time $\widetilde{O}(\epsilon^{-3})$. Unlike all other algorithms, this algorithm has two-sided error probability, which is unavoidable for testing this property (within $o(\sqrt{N})$ queries).

The complexity of Bipartiteness testing is considered in Section 3. We survey an $\Omega(\sqrt{N})$ lower bound on the query complexity of any tester [19] and a recent result of [20] by which a natural algorithm of running time (and query complexity) $\widetilde{O}(\mathrm{poly}(1/\epsilon) \cdot \sqrt{N})$ is a good tester. The lower bound stands in sharp contrast to the situation in the first model (i.e., representation by adjacency predicates), where Bipartite testing is possible in $\mathrm{poly}(1/\epsilon)$-time. We note that the $\widetilde{O}(\mathrm{poly}(1/\epsilon) \cdot \sqrt{N})$-time tester (for the incidence function representation) is significantly faster than the linear time decision procedure (provided that $\epsilon$ is not too small).

## 1.3 Combinatorics beyond Graph Theory – Testing Monotonicity

In this subsection we mention a partial result referring to testing a combinatorial property which is seemingly unrelated to graph theory. A function $f : \{0,1\}^n \mapsto \{0,1\}$ is called monotone if $f(x) \leq f(y)$, for every $x \prec y$, where the partial order between strings is defined analogously to the set inclusion relation. That is, $x_1x_2\cdots x_n \prec y_1y_2\cdots y_n$ if $x_i \leq y_i$ for all $i$'s, and $x \neq y$. Let $w(x)$ denote the *weight* of $x$ (i.e., the number of 1's in $x$). The following natural test of Monotonicity was suggested and analyzed in [18].

**Algorithm 1** (Monotonicity Tester):
*Given $n$, $\epsilon$ and oracle access to a function $f:\{0,1\}^n \mapsto \{0,1\}$, repeat $2n^2/\epsilon$ times:*

1. *Uniformly select $x \in \{0,1\}^n$, and obtain the value $f(x)$.*
2. *In case $f(x) = 1$, obtain the values of $f(y)$ for all $y$'s satisfying $y \succ x$ and $w(y) = w(x)+1$. If one of these $f(y)$'s is 0 then* reject.
3. *Analogously, in case $f(x) = 0$, obtain the values of $f(y)$ for all $y$'s satisfying $y \prec x$ and $w(y) = w(x)-1$. If one of these $f(y)$'s is 1 then* reject.

*If all iterations were completed without rejecting then* accept.

Clearly, Algorithm 1 accepts every monotone function with probability 1. Establishing the fact that it rejects, with probability at least 2/3, any function which is $\epsilon$-far from being monotone was reduced in [18] to the following combinatorial lemma referring to the Boolean Lattice (cf., background in [12]). For each $i$, $0 \leq i \leq n$, let $\mathrm{L}_i \subset \{0,1\}^n$ denote the set of $n$-bit long strings of weight $i$. Let $\mathrm{G}_n$ be the leveled *directed* (acyclic) graph over the vertex set $\{0,1\}^n$, where there is a directed edge from $y$ to $x$ if and only if $x \prec y$ and $w(x) = w(y)-1$ (i.e., $x$ and $y$ are in adjacent $L_i$'s).

**Lemma 1.1** [18]: *Let $r$ and $s$ be integers satisfying, $0 \leq r < s \leq n$, and let $\mathrm{R}, \mathrm{S} \subset \{0,1\}^n$, be sets such that $\mathrm{R} \subseteq \mathrm{L}_r$, and $\mathrm{S} \subseteq \mathrm{L}_s$, and $|\mathrm{R}| = |\mathrm{S}| = m$. Suppose that there exists a 1-to-1 mapping $\psi$ from $\mathrm{S}$ to $\mathrm{R}$ such that for every $y \in \mathrm{S}$, there is a directed path in $G_n$ from $y$ to $\psi(y)$. Then there exist $m$ vertex-disjoint directed paths from $\mathrm{S}$ to $\mathrm{R}$ in $\mathrm{G}_n$.*

We stress that these vertex-disjoint paths do not have to respect $\psi$. In fact, if one requires these paths to respect $\psi$ then the lemma becomes false (cf., [18]). Using Lemma 1.1 we have

**Theorem 1** *Algorithm 1 is a property tester for monotonicity.*

### 1.4 Rest of this survey

In Sections 2 and 3 we consider the complexity of testing Bipartiteness in the two graph representations discussed above: In Section 2 we consider representation by an adjacency predicate, and in Section 3 by an incidence function. Concluding remarks appear in Section 4.

## 2 Testing Bipartiteness in the First Representation

In this section we consider the representation of $N$-vertex graphs by adjacency predicates mapping pairs $\{1,2,...,N\} \times \{1,2,...,N\}$ to $\{0,1\}$. The bipartite tester is extremely simple: It selects a tiny, random set of vertices and checks whether the induced subgraph is bipartite.

**Algorithm 2** (Bipartite Tester in the first model [17]):
*On input $N$, $d$, $\epsilon$ and oracle access to an adjacency predicate of an $N$-vertex graph, $\mathrm{G} = (\mathrm{V}, \mathrm{E})$:*

1. *Uniformly select a subset of* $\widetilde{O}(1/\epsilon^2)$ *vertices of* V.

2. *Accept if and only if the subgraph induced by this subset is Bipartite.*

Step (2) amounts to querying the predicate on all pairs of vertices in the subset selected at Step (1). As will become clear from the analysis, it actually suffice to query only $\widetilde{O}(1/\epsilon^3)$ of these pairs.

**Theorem 2** [17]: *Algorithm 2 is a Bipartite Tester* (in the adjacency predicate representation). *Furthermore, the algorithm always accepts a Bipartite graph, and in case of rejection it provides a witness of length* $\mathrm{poly}(1/\epsilon)$ (that the graph is not bipartite).

**Proof:** Let R be the subset selected in Step (1), and $G_R$ the subgraph induced by it. Clearly, if G is bipartite then so is $G_R$, for any R. The point is to prove that if G is $\epsilon$-far from bipartite then the probability that $G_R$ is bipartite is at most 1/3. Thus, from this point on we assume that at least $\epsilon N^2$ edges have to be omitted from G to make it bipartite.

We view R as a union of two disjoint sets U and S, where $t \stackrel{\rm def}{=} |U| = O(\epsilon^{-1} \cdot \log(1/\epsilon))$ and $m \stackrel{\rm def}{=} |S| = O(t/\epsilon)$. We will consider all possible partitions of U, and associate a partial partition of V with each such partition of U. The idea is that in order to be consistent with a given partition, $(U_1, U_2)$, of U, all neighbors of $U_1$ (resp., $U_2$) must be placed opposite to $U_1$ (resp., $U_2$). We will show that, with high probability, most high-degree vertices in V do neighbor U and so are forced by its partition. Since there are relatively few edges incident to vertices which do not neighbor U, it follows that with very high probability each such partition of U is detected as illegal by $G_R$. Details follow, but before we proceed let us stress the key observation: It suffices to rule out relatively few (partial) partitions of V (i.e., these induced by partitions of U), rather than all possible partitions of V.

We use the notations $\Gamma(v) \stackrel{\rm def}{=} \{u : (u,v) \in E\}$ and $\Gamma(X) \stackrel{\rm def}{=} \cup_{v\in X}\Gamma(v)$. Given a partition $(U_1, U_2)$ of U, we define a (possibly partial) partition, $(V_1, V_2)$, of V so that $V_1 \stackrel{\rm def}{=} \Gamma(U_2)$ and $V_2 \stackrel{\rm def}{=} \Gamma(U_1)$ (assume, for simplicity that $V_1 \cap V_1$ is indeed empty). As suggested above, if one claims that G can be "bipartited" with $U_1$ and $U_2$ on different sides then $V_1 = \Gamma(U_1)$ must be on the opposite side to $U_2$ (and $\Gamma(U_1)$ opposite to $U_1$). Note that the partition of U places no restriction on vertices which have no neighbor in U. Thus, we first ensure that *most* "influential" (i.e., "high-degree") vertices in V have a neighbor in U.

**Definition 2.1** (high-degree vertices and good sets): *We say that a vertex* $v \in V$ *is of* high-degree *if it has degree at least* $\frac{\epsilon}{3}N$. *We call* U good *for* V *if all but at most* $\frac{\epsilon}{3}N$ *of the high-degree vertices in* V *have a neighbor in* U.

Note that NOT insisting that U neighbors *all* high-degree vertices allows us to show that a random U of size unrelated to the size of the graph has this feature. (If we were to insist that U neighbors *all* high-degree vertices then we would have had to use $|U| = \Omega(\log N)$.)

**Claim 2.2** *With probability at least* 5/6, *a uniformly chosen set* U *of size* $t$ *is good.*

**Proof:** For any high-degree vertex $v$, the probability that $v$ does not have any neighbor in a uniformly chosen U is at most $(1-\epsilon/3)^t < \frac{\epsilon}{18}$ (since $t = \Omega(\epsilon^{-1}\log(1/\epsilon))$). Hence the expected number of high-degree vertices which do not have a neighbor in a random set U is less than $\frac{\epsilon}{18} \cdot N$, and the claim follows by Markov's Inequality. □

**Definition 2.3** (disturbing a partition of U): *We say that an edge* disturbs *a partition* $(U_1, U_2)$ *of* U *if both is end-points are in the same* $\Gamma(U_i)$, *for some* $i \in \{1, 2\}$.

**Claim 2.4** *For any good set* U *and any partition of* U*, at least* $\frac{\epsilon}{3}N^2$ *edges disturb the partition.*

**Proof:** Each partition of V has at least $\epsilon N^2$ violating edges (i.e., edges with both end-points on the same side). We upper bound the number of these edges which are not disturbing. Actually, we upper bound the number of edges which have an end-point not in $\Gamma(\mathrm{U})$.

- The number of edges incident to high-degree vertices which do not neighbor U is bounded by $\frac{\epsilon}{3}N \cdot N$ (at most $\frac{\epsilon}{3}N$ such vertices each having at most $N$ incident edges).
- The number of edges incident to vertices which are not of high-degree vertices is bounded by $N \cdot \frac{\epsilon}{3}N$ (at most $N$ such vertices each having at most $\frac{\epsilon}{3}N$ incident edges).

This leaves us with at least $\frac{\epsilon}{3}N^2$ violating edges connecting vertices in $\Gamma(\mathrm{U})$ (i.e., edges disturbing the partition of U). $\square$

The theorem follows by observing that $\mathrm{G_R}$ is bipartite only if either (1) the set U is not good; or (2) the set U is good and there exists a partition of U so that none of the disturbing edges occurs in $\mathrm{G_R}$. Using Claim 2.2 the probability of event (1) is bounded by 1/6; and using Claim 2.4 the probability of event (2) is bounded by the probability that there exists a partition of U so that none of the corresponding $\geq \frac{\epsilon}{3}N^2$ disturbing edges has both edge-point in S. Actually, we pair the $m$ vertices of S, and consider the probability that none of these pairs is a disturbing edge for a partition of U. Thus the probability of event (2) is bounded by

$$2^{|\mathrm{U}|} \cdot \left(1 - \frac{\epsilon}{3}\right)^{m/2} < \frac{1}{6}$$

where the inequality is due to $m = \Omega(t/\epsilon)$. The theorem follows. $\blacksquare$

**Comment:** The procedure employed in the proof yields a poly$(1/\epsilon) \cdot N$-time algorithm for 2-partitioning a bipartite graph so that at most $\epsilon N^2$ edges lie within the same side. This is done by running the tester, determining a partition of U (defined as in the proof) which is consistent with the bipartite partition of R, and partitioning V as done in the proof (with vertices which do not neighbor U, or neighbor both $\mathrm{U}_1, \mathrm{U}_2$, placed arbitrarily). Thus, the placement of each vertex is determined by inspecting at most $\widetilde{O}(1/\epsilon)$ entries of the adjacency matrix.

# 3 Testing Bipartiteness in the Second Representation

In this section we consider the representation of $N$-vertex graphs of degree bound $d$ by incidence functions mapping pairs $\{1, 2, ..., N\} \times \{1, 2, ..., d\}$ to $\{0, 1, 2, ..., N\}$.

## 3.1 A lower bound

In contrast to Theorem 2, under the incidence function representation there exists no Bipartite tester of complexity independent of the graph size.

**Theorem 3** [19]: *Testing Bipartiteness* (with constant $\epsilon$ and $d$) *requires* $\Omega(\sqrt{N})$ *queries* (in the incidence function representation).

**Proof Idea:** For any (even) $N$, we consider the following two families of graphs:

1. The first family, denoted $\mathcal{G}_1^N$, consists of all degree-3 graphs which are composed by the union of a Hamiltonian cycle and a perfect matching. That is, there are $N$ edges connecting the vertices in a cycle, and the other $N/2$ edges are a perfect matching.

2. The second family, denoted $\mathcal{G}_2^N$, is the same as the first *except* that the perfect matchings allowed are restricted as follows: the distance on the cycle between every two vertices which are connected by an perfect matching edge must be odd.

Clearly, all graphs in $\mathcal{G}_2^N$ are bipartite. One first proves that almost all graphs in $\mathcal{G}_1^N$ are far from being bipartite. Afterwards, one proves that a testing algorithm that performs less than $\alpha\sqrt{N}$ queries (for some constant $\alpha < 1$) is not able to distinguish between a graph chosen randomly from $\mathcal{G}_2^N$ (which is always bipartite) and a graph chosen randomly from $\mathcal{G}_1^N$ (which with high probability will be far from bipartite). Loosely speaking, this is done by showing that in both cases the algorithm is unlikely to encounter a cycle (among the vertices it has inspected). ■

### 3.2 An algorithm

The lower bound of Theorem 3 is essentially tight. Furthermore, the following natural algorithm constitutes a Bipartite tester of running time $\text{poly}((\log N)/\epsilon) \cdot \sqrt{N}$.

**Algorithm 3** (Bipartite Tester in the second model [20]):
*On input $N$, $d$, $\epsilon$ and oracle access to an incidence function for an $N$-vertex graph,* $\mathrm{G} = (\mathrm{V}, \mathrm{E})$*, of degree bound $d$, repeat $T \stackrel{\text{def}}{=} \Theta(\frac{1}{\epsilon})$ times:*

1. *Uniformly select $s$ in* $\mathrm{V}$.
2. (Try to find an odd cycle through vertex $s$):
   (a) *Perform $K \stackrel{\text{def}}{=}$* $\text{poly}((\log N)/\epsilon) \cdot \sqrt{N}$ *random walks starting from $s$, each of length* $L \stackrel{\text{def}}{=} \text{poly}((\log N)/\epsilon)$.
   (b) *Let $R_0$* (resp., $R_1$) *denote the vertices set reached from $s$ in an even* (resp., odd) *number of steps in any of these walks.*
   (c) *If $R_0 \cap R_1$ is not empty then* reject.

*If the algorithm did not reject in any one of the above $T$ iterations, then it* accepts.

**Theorem 4** [20]: *Algorithm 3 is a Bipartite Tester* (in the incidence function representation). *Furthermore, the algorithm always accepts a Bipartite graph, and in case of rejection it provides a witness of length* $\text{poly}((\log N)/\epsilon)$ (that the graph is not bipartite).

**Motivation – the special case of rapid mixing graphs.** The proof of Theorem 4 is quite involved. As a motivation, we consider the special case where the graph has a "rapid mixing" feature. It is convenient to modify the random walks so that at each step each neighbor is selected with probability $1/2d$, and otherwise (with probability at least $1/2$) the walk remains in the present vertex. Furthermore, we will consider a single execution of Step (2) starting from an arbitrary vertex, $s$, fixed in the rest of the discussion. The rapid mixing feature we assume is that, for every vertex $v$, a (modified) random walk of length $L$ starting at $s$ reaches $v$ with probability approximately $1/N$ (say, up-to a factor of 2). Note that if the graph is an expander then this is certainly the case (since $L \geq O(\log N)$).

The key quantities is the analysis are the following probabilities, referring to the parity of the length of a path obtained from the random walk by omitting the self-loops (transitions which remain at current vertex). Let $p^0(v)$ (resp., $p^1(v)$) denote the probability that a (modified) random walk of length $L$ starting at $s$ reaches $v$ while making an even (resp., odd) number of real (i.e., non-self-loop) steps. By the rapid mixing assumption we have (for every $v \in \mathrm{V}$)

$$\frac{1}{2N} \; < \; p^0(v) + p^1(v) \; < \; \frac{2}{N} \tag{1}$$

We consider two cases regarding the sum $\sum_{v \in \mathrm{V}} p^0(v)p^1(v)$ – In case the sum is (relatively) "small", we show that V can be 2-partitioned so that there are relatively few edges between vertices placed in the same side, which implies that G is close to be bipartite. Otherwise (i.e., when the sum is not "small"), we show that with significant probability, when Step (2) is started at vertex $s$ it is completed by rejecting G. The two cases are presented in greater detail in the following (corresponding) two claims.

**Claim 3.1** *Suppose* $\sum_{v \in \mathrm{V}} p^0(v)p^1(v) \;\le\; \epsilon/50N$. *Let* $\mathrm{V}_1 \stackrel{\mathrm{def}}{=} \{v \in \mathrm{V} : p^0(v) < p^1(v)\}$ *and* $\mathrm{V}_2 = \mathrm{V} \setminus \mathrm{V}_1$. *Then, the number of edges with both end-points in the same* $\mathrm{V}_\sigma$ *is bounded above by* $\epsilon dN$.

**Proof Sketch:** Consider an edge $(u, v)$ where, without loss of generality, both $u$ and $v$ are in $\mathrm{V}_1$. Then, both $p^1(v)$ and $p^1(u)$ are greater than $\frac{1}{2} \cdot \frac{1}{2N}$. However, one can show that $p^0(v) > \frac{1}{3d} \cdot p^1(u)$: Observe that a walk of length $L - 1$ with path-parity 1 ending at $u$ is almost as likely as such a walk having length $L$, and that once such a walk reaches $u$ it continues to $v$ in the next step with probability exactly $1/2d$. Thus, such an edge contributes at least $\frac{(1/4N)^2}{3d}$ to the sum $\sum_{v \in \mathrm{V}} p^0(v)p^1(v)$. The claim follows. ■

**Claim 3.2** *Suppose* $\sum_{v \in \mathrm{V}} p^0(v)p^1(v) \;\ge\; \epsilon/50N$, *and that Step (2) is started with vertex* $s$. *Then, with probability at least* $2/3$, *the set* $R_0 \cap R_1$ *is not empty* (and rejection follows).

**Proof Sketch:** Consider the probability space defined by an execution of Step (2) with start vertex $s$. We define random variables $\zeta_{i,j}$ representing the event that the vertex encountered in the $L^{\text{th}}$ step of the $i^{\text{th}}$ walk equals the vertex encountered in the $L^{\text{th}}$ step of the $j^{\text{th}}$ walk, and that the $i^{\text{th}}$ walk corresponds to an even-path whereas the $j^{\text{th}}$ to an odd-path. Then

$$\begin{aligned}
\mathbf{E}(|R_0 \cap R_1|) &> \sum_{i \neq j} \mathbf{E}(\zeta_{i,j}) \\
&= K(K-1) \cdot \sum_{v \in \mathrm{V}} p^0(v)p^1(v) \\
&> \frac{500N}{\epsilon} \cdot \sum_{v \in \mathrm{V}} p^0(v)p^1(v) \\
&\ge 10
\end{aligned}$$

where the second inequality is due to the setting of $K$, and the third to the claim's hypothesis. Intuitively, we expect that with high probability $|R_0 \cap R_1| > 0$. This is indeed the case, but proving it is less straightforward than it seems, the problem being that the $\zeta_{i,j}$'s are not pairwise independent. Yet, since the sum of the covariances of the dependent $\zeta_{i,j}$'s is quite small, Chebyshev's Inequality is still very useful (cf., [1, Sec. 4.3]). Specifically, letting $\mu \stackrel{\mathrm{def}}{=} \sum_{v \in \mathrm{V}} p^0(v)p^1(v)$ $(= \mathbf{E}(\zeta_{i,j}))$, and

$\overline{\zeta}_{i,j} \stackrel{\rm def}{=} \zeta_{i,j} - \mu$, we get:

$$\begin{aligned}
\mathbf{Pr}\left(\sum_{i\neq j} \zeta_{i,j} = 0\right) &< \frac{\mathbf{V}(\sum_{i\neq j} \zeta_{i,j})}{(K^2\mu)^2} \\
&= \frac{1}{K^4\mu^2} \cdot \left(\sum_{i,j} \mathbf{E}(\overline{\zeta}_{i,j}^2) + 2\sum_{i,j,k} \mathbf{E}(\overline{\zeta}_{i,j}\overline{\zeta}_{i,k})\right) \\
&< \frac{1}{K^2\mu} + \frac{2}{K\mu^2} \cdot \mathbf{E}(\zeta_{1,2}\zeta_{1,3})
\end{aligned}$$

For the second term, we observe that $\mathbf{Pr}(\zeta_{1,2} = \zeta_{2,3} = 1)$ is upper bounded by the probability that $\zeta_{1,2} = 1$ times the probability that the $L^{\rm th}$ vertex of the first walk appears as the $L^{\rm th}$ vertex of the third path. Using the rapid mixing hypothesis, we upper bound the latter probability by $2/N$, and obtain

$$\begin{aligned}
\mathbf{Pr}(|R_0 \cap R_1| = 0) &< \frac{1}{K^2\mu} + \frac{2}{K\mu^2} \cdot \mu \cdot \frac{2}{N} \\
&< \frac{1}{3}
\end{aligned}$$

where the last inequality uses $K < N/4$, $\mu \geq \epsilon/50N$ and $K^2 \geq 6 \cdot 50N/\epsilon$. The claim follows. ■

**Beyond rapid mixing graphs.** The proof in [20] refers to a more general sum of products; that is, $\sum_{v\in \mathrm{V}} p_{\rm odd}(v)p_{\rm even}(v)$, where $\mathrm{U} \subseteq \mathrm{V}$ is an appropriate set of vertices, and $p_{\rm odd}(v)$ (resp., $p_{\rm even}(v)$) is the probability that a random walk (starting at $s$) passes through $v$ after more than $L/2$ steps and the corresponding path to $v$ has odd (resp., even) parity. Much of the analysis in [20] goes into selecting the appropriate U (and an appropriate starting vertex $s$), and pasting together many such U's to cover all of V. Loosely speaking, U and $s$ are selected so that there are few edges from U and the rest of the graph, and $p_{\rm odd}(u) + p_{\rm even}(u) \approx 1/\sqrt{|\mathrm{V}| \cdot |\mathrm{U}|}$, for every $u \in \mathrm{U}$. The selection is based on the "combinatorial treatment of expansion" of Mihail [31]. Specifically, we use the counterpositive of the standard analysis, which asserts that rapid mixing occurs when all cuts are relatively large, to assert the existence of small cuts which partition the graph so that vertices reached with relatively high probability (in a short random walk) are on one side and the rest of the graph on the other. The first set corresponds to U above and the cut is relatively small with respect to U. A start vertex $s$ for which the corresponding sum is big is shown to cause Step (2) to reject (when started with this $s$), whereas a small corresponding sum enables to 2-partition U while having few violating edges among the vertices in each part of U.

The actual argument of [20] proceeds in iterations. In each iteration a vertex $s$ for which Step (2) accepts with high probability is fixed, and an appropriate set of remaining vertices, U, is found. The set U is then 2-partitioned so that there are few violating edges inside U. Since we want to paste all these partitions together, U may not contain vertices treated in previous iterations. This complicates the analysis, since it must refer to the part of G, denoted H, not treated in previous iterations. We consider walks over an (imaginary) Markov Chain representing the H-part of the walks performed by the algorithm on G. Statements about rapid mixing are made with respect to the Markov Chain, and linked to what happens in random walks performed on G. In particular, a subset U of H is determined so that the vertices in U are reached with probability $\approx 1/\sqrt{|\mathrm{V}| \cdot |\mathrm{U}|}$ (in the chain) and the cut between U and the rest of H is small. Linking the sum of products defined for the chain with the actual walks performed by the algorithm, we infer that U may be partitioned with few violating edges inside it. Edges to previously treated parts of the graphs are charged to these parts, and edges to the rest of $H \setminus U$ are accounted for by using the fact that this cut is small (relative to the size of U).

# 4 Concluding Remarks

Randomness plays a pivotal role in the theory of property testing: A deterministic tester for any "non-degenerate" property (and in particular for any of the properties discussed above) needs to query the function on a constant fraction of its domain, and so is of little interest.

The results regarding property testing, known to date, are rather sporadic. For more than a dozen natural graph properties, testers are known in the adjacency predicate representation, and for some testers are known in the incidence function representation. More than half a dozen of these testers are interesting, and though they share some techniques, no general structure seems to arise. Some negative results in [17] seem to indicate that general results may be hard to obtain: For example, it was shown that there exist properties in $\mathcal{NP}$ which require high query complexity for testing. Also some properties are easy to test with one-sided error, whereas other require two-sided error to be tested efficiently. Thus, obtaining "structural" results regarding easily testable properties may be very challenging as well as of great interest.


# Acknowledgments

This survey is based mainly on joint work with Shafi Goldwasser and Dana Ron [17], and joint works with Dana Ron [19, 20]. The subsection on testing monotonicity is based on a recent joint work with Shafi Goldwasser, Eric Lehman, and Dana Ron [18].


# References


[1] N. Alon and J.H. Spencer, *The Probabilistic Method*, John Wiley & Sons, Inc., 1992.

[2] S. Arora, D. Karger, and M Karpinski. Polynomial time approximation schemes for dense instances of NP-hard problems. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing*, pages 284–293, 1995.

[3] S. Arora, C. Lund, R. Motwani, M. Sudan, and M. Szegedy. Proof verification and intractability of approximation problems. In *Proceedings of the Thirty-Third Annual Symposium on Foundations of Computer Science*, pages 14–23, 1992.

[4] S. Arora and S. Safra. Probabilistic checkable proofs: A new characterization of NP. In *Proceedings of the Thirty-Third Annual Symposium on Foundations of Computer Science*, pages 1–13, 1992.

[5] M. Bellare, D. Coppersmith, J. Håstad, M. Kiwi, and M. Sudan. Linearity testing in characteristic two. In *Proceedings of the Thirty-Sixth Annual Symposium on Foundations of Computer Science*, pages 432–441, 1995.

[6] L. Babai, L. Fortnow, and C. Lund. Non-deterministic exponential time has two-prover interactive protocols. *Computational Complexity*, 1(1):3–40, 1991.

[7] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 21–31, 1991.

[8] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs and applications to approximation. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, pages 294–304, 1993.

[9] M. Bellare, O. Goldreich, and M. Sudan. Free bits, pcps and non-approximability – towards tight results. Extended abstract in *Proceedings of the Thirty-Sixth Annual Symposium on Foundations of Computer Science*, pages 422–431, 1995. To appear in *SICOMP*.

[10] M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47:549–595, 1993.

[11] M. Bellare and M. Sudan. Improved non-approximability results. In *Proceedings of the 26th Annual ACM Symposium on the Theory of Computing*, pages 184–193, 1994.

[12] B. Bollobás. *Combinatorics*. Cambridge University Press, 1986.

[13] W. F. de la Vega. MAX-CUT has a randomized approximation scheme in dense graphs. To appear in *Random Structures and Algorithms*, 1994.

[14] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating clique is almost NP-complete. In *Proceedings of the Thirty-Second Annual Symposium on Foundations of Computer Science*, pages 2–12, 1991.

[15] A. Frieze and R. Kanan. The regularity lemma and approximation schemes for dense problems. In *Proceedings of the Thirty-Seventh Annual Symposium on Foundations of Computer Science*, pages 12–20, 1996.

[16] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the Twenty-Third Annual ACM Symposium on Theory of Computing*, pages 32–42, 1991.

[17] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. To appear in *JACM*. Extended abstract in *Proc. of the 37th IEEE Symp. on Foundation of Computer Science*, pages 339–348, 1996. Available as TR96-057 of *ECCC*, `http://www.eccc.uni-trier.de/eccc/`, 1996.

[18] O. Goldreich, S. Goldwasser, E. Lehman, and D. Ron. Testing Monotinicity. Unpublished manuscript, 1998.

[19] O. Goldreich and D. Ron. Property Testing in Bounded Degree Graphs. In *Proc. of the 29th ACM Symp. on Theory of Computing*, pages 406–415, 1997.

[20] O. Goldreich and D. Ron. A sublinear Bipartite Tester for Bounded Degree Graphs. To appear in *Proc. of the 30th ACM Symp. on Theory of Computing*, 1998. Available from `http://theory.lcs.mit.edu/~oded/test.html`.

[21] P. Hajnal. An $\Omega(n^{4/3})$ lower bound on the randomized complexity of graph properties. *Combinatorica*, 11(2):131–144, 1991.

[22] J. Håstad. Testing of the long code and hardness for clique. In *Proc. of the 28th ACM Symp. on Theory of Computing*, pages 11–19, 1996.

[23] J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Proc. of the 37th IEEE Symp. on Foundation of Computer Science*, pages 627–636, 1996.

[24] J. Håstad. Getting optimal in-approximability results. In *Proc. of the 29th ACM Symp. on Theory of Computing*, pages 1–10, 1997.

[25] D. S. Hochbaum and D. B. Shmoys. Using dual approximation algorithms for scheduling problems: Theoretical and practical results. *Journal of the Association for Computing Machinery*, 34(1):144–162, January 1987.

[26] D. S. Hochbaum and D. B. Shmoys. A polynomial approximation scheme for machine scheduling on uniform processors: Using the dual approximation approach. *SIAM Journal on Computing*, 17(3):539–551, 1988.

[27] V. King. An $\Omega(n^{5/4})$ lower bound on the randomized complexity of graph properties. *Combinatorica*, 11(1):23–32, 1991.

[28] M. Kiwi. *Probabilistically Checkable Proofs and the Testing of Hadamard-like Codes*. PhD thesis, Massachusetts Institute of Technology, 1996.

[29] R. J. Lipton. New directions in testing. Unpublished manuscript, 1989.

[30] L. Lovász and N. Young. Lecture notes on evasiveness of graph properties. Technical Report TR–317–91, Princeton University, Computer Science Department, 1991.

[31] M. Mihail. Conductance and convergence of Markov chains – A combinatorial treatment of expanders. In *Proceedings 30th Annual Symp. on Foundations of Computer Science*, pages 526–531, 1989.

[32] A. L. Rosenberg. On the time required to recognize properties of graphs: A problem. *SIGACT News*, 5:15–16, 1973.

[33] R. Rubinfeld. Robust functional equations and their applications to program testing. In *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 1994. To appear in *SIAM Journal on Computing*.

[34] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal on Computing*, 25(2):252–271, 1996.

[35] R. L. Rivest and J. Vuillemin. On recognizing graph properties from adjacency matrices. *Theoretical Computer Science*, 3:371–384, 1976.

[36] L. Trevisan. Recycling queries in PCPs and in linearity tests. To appear in *Proc. of the 30th ACM Symp. on Theory of Computing*, 1998.

[37] L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.

[38] A. C. C. Yao. Lower bounds to randomized algorithms for graph properties. In *Proceedings of the Twenty-Eighth Annual Symposium on Foundations of Computer Science*, pages 393–400, 1987.