Local Property Reconstruction and Monotonicity[*]

Michael Saks[†]

saks@math.rutgers.edu
Dept. of Mathematics
Rutgers University

C. Seshadhri[‡]

csesha@us.ibm.com
IBM Almaden Research Center

Abstract

We propose a general model of local property reconstruction. Suppose we have a function f on domain Γ, which is supposed to have a particular property $\mathcal{P}$, but may not have the property. We would like a procedure which produces a function g that has property $\mathcal{P}$ and is close to f (according to some suitable metric). The reconstruction procedure, called a *filter*, has the following form. The procedure takes as input an element x of Γ and outputs $g(x)$. The procedure has oracle access to the function f and uses a single short random string ρ, but is otherwise deterministic.

This model was inspired by a related model of online property reconstruction that was introduced by by Ailon, Chazelle, Comandur and Liu (2004). It is related to the property testing model, and extends the framework that is used in the model of locally decodable codes. A similar model, in the context of hypergraph properties, was independently proposed and studied by Austin and Tao (2008).

We specifically consider the property of monotonicity and develop an efficient local filter for thie property. The input f is a real valued function defined on the domain $\{1, \ldots, n\}^d$ (where n is viewed as large and d as a constant). The function is monotone if the following property holds: for two domain elements x and y, if $x \leq y$ (in the product order) then $f(x) \leq f(y)$. Given x, our filter outputs the value $g(x)$ in $(\log n)^{O(1)}$ time and uses a random seed ρ of the same size. With high probability, the ratio of the (Hamming) distance between g and f to the minimum possible Hamming distance between a monotone function and f is bounded above by a function of d (independent of n).

[*]This is an extended abstract of work that will appear as "Local Monotonicity Reconstruction" in SIAM Journal of Computing. A preliminary version of this work appeared as "Parallel Monotonicity Reconstruction" [29].

[†]This work was supported in part by NSF under grants CCF-0515201 and CCF-0832787.

[‡]This paper is partly based on material that appeared in this author's Ph.D. dissertation for the Department of Computer Science, Princeton University.

1 Online property reconstruction

The process of assembling large data sets is prone to varied sources of corruption, such as measurement error, replication error, and communication noise. Error correction techniques (i.e. coding) can be used to reduce or eliminate the effects of some sources of error, but often some residual errors may be unavoidable. Despite the presence of such inherent error, the data set may still be very useful.

One problem in using such a data set is that even small amounts of error can significantly change the behavior of algorithms that act on the data. For example, if we do a binary search on an array that is supposed to be sorted, a few erroneous entries may lead to behavior that deviates significantly from the "correct" behavior.

This is an example of a more general situation. We have a data set that ideally should have some specified structural property, i.e., a list of numbers that should be sorted, a set of points that should be in convex position, or a graph that should be a tree. Algorithms that run on the data set may rely on this property. A small amount of error may destroy the property, and result in the algorithm producing wildly unexpected results, or even crashing. In these situations, a small amount of error may be tolerable but only if the structural property is maintained.

These considerations motivated the formulation of the *online property reconstruction* model, which was introduced in [3]. We are given a data set, which we think of as a function f defined on some domain Γ. Ideally, f should have a specified structural property $\mathcal{P}$, but this property may not hold due to unavoidable errors. We wish to construct *online* a new data set g such that:

(1) g has property $\mathcal{P}$ and (2) $d(g,f)$ is small, where $d(g,f)$ is the fraction of values $x \in \Gamma$ for which $g(x) \neq f(x)$.

How small should $d(g,f)$ be in Condition (2)? Define $\varepsilon_f = \varepsilon_f(\mathcal{P})$ to be the minimum of $d(h,f)$ over all h that satisfy $\mathcal{P}$. Of course, ε_f is a lower bound on the deviation of g from f. The *error blow-up* of g is the ratio $d(g,f)/\varepsilon_f$. This error blow-up can be viewed as the price that is paid in order to restore the property $\mathcal{P}$ online, and we want this to be a not too large constant.

An offline reconstruction algorithm explicitly outputs such a g on input f. In the context of large data sets, the explicit construction of g from f requires a considerable amount of computational overhead (at least linear in the size of the data set). For this reason, [3] considered online reconstruction algorithms. Such an algorithm, called a *filter*, gets as input a sequence $x_1, x_2, \ldots$ of elements of Γ presented one at a time and must output the sequence of values $g(x_1), g(x_2), \ldots$ where $g(x_i)$ is produced in response to x_i, before knowing x_{i+1}. The filter can access the function f via an oracle which, given $y \in \Gamma$, answers $f(y)$. The aim is to design a filter that, for any online input sequence of elements in Γ, outputs a function g satisfying (1) and (2) above and furthermore produces each successive $g(x_i)$ quickly, i.e., in time much smaller than $O(|\Gamma|)$.

In [3], a filter for the *monotonicity property* was given. In this setting, the domain Γ is the set $[n]^d = \{(j_1, \ldots, j_d) : j_i \in [n]\}$, where $[n]$ denotes the set $\{1, 2, \ldots, n\}$. The set $[n]^d$ is considered to be partially ordered under the component-wise (product) order: $(i_1, \ldots, i_d) \leq (j_1, \ldots, j_d)$ iff $\forall r, i_r \leq j_r$. A function f defined on Γ is *monotone* if $x \leq y$ implies $f(x) \leq f(y)$. The filter they constructed satisfies Condition (1), has error blow-up that is bounded above by $2^{O(d)}$ (independent of n), and answers each successive query in time $(\log n)^{O(d)}$.

2 Local property reconstruction

The filter for monotonicity proposed in [3] has the following general structure. For each successive query x_j, the filter executes a randomized algorithm to compute $g(x_j)$. This algorithm accesses f, and also needs to access the answers $g(x_i)$ for $i < j$ to the queries asked previously. In particular, the function g produced may depend on both the order of the queries and the random bits used by the algorithm.

This general structure for filters has two potential drawbacks: (1) It requires the storage of all previous queries and answers, thus incurring possibly significant space overhead for the algorithm, (2) It does not support a local implementation in which multiple copies of the filter, having read-only access to f, are able to handle queries independently while maintaining mutual consistency.

In this paper, we propose the following strengthened requirements for a filter. A *local filter*[1] for reconstructing property $\mathcal{P}$ is an algorithm A that has oracle access to a function f on domain Γ (the "data set") and to an auxiliary random string ρ (the "random seed"), and takes as input $x \in \Gamma$. For fixed f and ρ, A runs deterministically on input x to produce an output $A_{f,\rho}(x)$. Thus, given f and ρ, $A_{f,\rho}$ specifies a function on domain Γ. We want A to satisfy the following properties:

1. For each f and ρ, $A_{f,\rho}$ satisfies $\mathcal{P}$. [2]
2. For each f, with high probability (with respect to the choice of ρ), the function $A_{f,\rho}$ should be "suitably close" to f.
3. For each x, $A_{f,\rho}$ on x can be computed very quickly.
4. The size of the random seed ρ should be "much smaller" than $|\Gamma|$.

Remark 1: In Condition 2, we say that $A_{f,\rho}$ should be "suitably close" to f. There are various ways to make this precise. Let ε_f denote the minimum distance from f to a function satisfying $\mathcal{P}$ and let $\gamma_f(\rho)$ denote the distance from f to $A_{f,\rho}$. We would like $\gamma_f(\rho)$ to be small compared to ε_f. The error blow-up, which is the ratio of $\gamma_f(\rho)/\varepsilon_f$, works well for the monotoncity property that we study. For other properties, it might be more appropriate to use another criterion: for example, we might consider the difference $\gamma_f(\rho) - \varepsilon_f$. More generally, we could require simply that $\gamma_f(\rho)$ be bounded above by some arbitrary function of ε_f (either independent of $|\Gamma|$ or growing very slowly with $|\Gamma|$).

Remark 2: Similarly, for Condition 3, there are various possibilities for interpreting the phrase "very quickly". In this paper, we obtain running times that are polynomial in $\log |\Gamma|$. In Section 3, we will mention some work on other properties where the running time does not depend on the domain size. On the other hand, there may be other properties where it is non-trivial and interesting to obtain running times of the form $|\Gamma|^\delta$.

Remark 3: A local filter can be used, trivially, as an online filter. The space required by the local filter is bounded by the sum of the length of ρ and the running time per query. By keeping these both small (e.g., much smaller than $|\Gamma|$) we obtain an online filter using little auxiliary space.

Remark 4: A local filter can be used to enforce consistent behavior among autonomous processors who each have access to f but do not communicate with each other. We generate one random seed ρ and give the same random seed to each of the processors. Since $A_{f,\rho}$ is deterministic, all processors will reconstruct the same function.

[1]This was originally called a *parallel filter* in the conference version [29]. We made this terminology change since it is more compatible with the existing concepts of locally decodable codes.

[2]In an earlier version of this paper, this condition was replaced by the weaker condition that for each f, $A_{f,\rho}$ should satisfy $\mathcal{P}$ with high probability. Prompted by a question raised by a referee we were able to modify our monotonicity filter to satisfy this stronger property, and so modified the definition accordingly. The weaker condition may be more appropriate for some other properties.

3 Related Work

One case of property reconstruction that has been studied extensively is *error correcting codes*. Suppose $C \subseteq \{0,1\}^n$ is such a code in which all members of C are pairwise at distance at least d. Let $\mathcal{P}$ be the property of being a codeword. The error correction problem for C is to find the *closest codeword* to a given input string x. This can be formulated as a reconstruction problem for the property $\mathcal{P}$.

One variant of the error correction is the problem of local decoding. This problem was explicitly named in [25], but, as noted there, was studied previously in connection with self-correcting computation (e.g., [12, 20]), probabilistically checkable proofs (e.g., [8]), average-case reductions (e.g., [9, 30]), and private information retrieval (e.g., [13]). Here we want a decoding algorithm for a given code that, given oracle access to the bits of an input string x, and given an index $i \in [n]$, finds the ith bit of the closest codeword to x by querying a small (possibly randomly selected) number of bits of x. If we view the local decoding algorithm as a deterministic algorithm that takes input i and a random string r (used to make the decisions) then we require that for each i, most choices of r lead to the correct value for the ith bit of the closest codeword.

This is very similar to (though not quite the same as) the local property reconstruction problem for $\mathcal{P}$; for local property reconstruction we interchange the "for all" and "for most" quantifiers and require that for most choices of r, and for all $i \in [n]$, the algorithm correctly produces the ith bit of the codeword. Also, we pay attention to the length of the random string r, which we want to be suitably small.

In local list decoding, our aim is to find a short *list* of codewords that are all suitably close to the input word. For example, in list decoding of low-degree polynomials [6, 30], the input is a function and the output is a *small list* of low-degree polynomials that are close to the input function.

The monotonicity problem considered in this paper is qualitatively quite different from the local decoding examples. In local decoding there is either *one* correct output, or (in the case of list-decoding) a *sparse list* of possible correct outputs. For monotonicity there may be many (possibly infinitely many) ways to correct a given function to a nearby function with the desired property. One might think that having many possible close corrections (rather than one) makes reconstruction easier but, at least for the monotonicity problem, it does not. The difficulty arises from the requirement that once the random seed is fixed, all query answers provided by the filter must be consistent with a *single* function having the property.

A related notion of reconstruction was discussed in [23], for generalized partition problems in dense graphs. Given an input dense graph G that satisfies some partition property (say k-colorability), we wish to efficiently construct a partition of the vertices that has at most an ε-fraction of violating edges. The algorithms for this problem provided in [23] behaved like local filters. Specifically, there was a constant (function of ε) time algorithm that gave the color class of an input vertex of G, and this could be run independently on all vertices (after fixing a random seed). This coloring was guaranteed to violate at most an ε-fraction of the edges in G.

Independently of our work, a model of *repair* of a property was formulated and studied in [7]. This is closely related to the reconstruction model considered here. The results in [7] primarily considered reconstruction of *hypergraph properties*, and obtained local filters of a very special form that modify an input hypergraph to satisfy a given property. This result can be seen as a generalization of the characterizations of testable properties of dense graphs [4, 5]. This does not focus on the exact form of the error blow-up, and only requires that the distance of the reconstructed hypergraph be bounded by some arbitrary function of the minimum distance of the hypergraph to the property.

In general, a local filter for reconstructing a given property can be used to estimate the distance of an input instance to the property. When we fix a random seed and run the filter on f, the filter implicitly outputs a function g that has the desired property and is at distance at most $B\varepsilon_f$ from f (where ε_f is distance of f to $\mathcal{P}$). By choosing a random sample of domain points x and computing the fraction of points where $g(x) \neq f(x)$, we get an estimate of the distance $d(g, f)$. Since $\varepsilon_f \leq d(g, f)$ and with high probability, $\varepsilon_f \geq d(g, f)/B$, we get a multiplicative B-approximation to ε_f in sublinear time.

4 Our results

In this work, we construct a local filter for monotonicity for functions defined on $[n]^d$ with the following performance:

- The time per query is $(\log n)^{O(d)}$.
- The error blow-up is $2^{O(d^2)}$, independent of n.
- The number of random bits needed to initialize the filter is $(d \log n)^{O(1)}$.

The online filter for monotonicity of [3] has a running time per query of $(\log n)^{O(d)}$ (with a better constant in the exponent) and an error blow-up of 2^d. We see that our filter achieves local behavior while having query time and error blow-up that are similar to (but not quite as good) as those obtained by [3].

Our filter for monotonicity builds on techniques used for property testing of monotonicity. There has been a large amount of work done on property testing, which was defined in [23, 28]. Many testers have been given for a wide variety of combinatorial, algebraic, and geometric problems (see surveys [17, 21, 27]). The related notions of tolerant testing and distance approximation were introduced in [26]. The problem of monotonicity in the context of property testing has been studied in [1, 10, 11, 14, 15, 18, 19, 22, 24]. Sublinear algorithms for *approximating* the distance of a function to monotonicity have been given in [2, 16, 26].

Both the running time and error blow-up of our filter have an exponential dependence on the dimension d. We also prove that this dependence is unavoidable. Specifically we show the following for some constant $0 < \alpha < 1$: given a filter on the boolean hypercube $\{0, 1\}^d$ that answers queries within time $2^{\alpha d}$, there is an input function f such that the filter applied to f has error blow-up $2^{\alpha d}$ with probability close to $1/2$. This shows a complexity gap between testing and reconstruction for the hypercube, since there are monotonicity testers with only a polynomial dependence on d [14, 16, 22].

5 Overview of the local filter for monotonicity

We now discuss some of the ideas used in constructing the looal filter for monotonicity. Details of the construction and analysis can be found in the full paper.

The starting point for the construction of our local filter for monotonicity is the online filter of [3]. We now give the main ideas of their construction, and indicate the difficulties in making their construction local. In the discussion below, when we say an algorithm is "fast", we mean that it runs in time polylogarithmic in $|\Gamma|$.

We start with the case $d = 1$, i.e., the one-dimensional case. The basic idea (implicitly used) in [3] is to classify the domain points as *accepted* and *rejected* in such a way that the following conditions hold:
(1) There is a fast algorithm for testing whether a given point is accepted or rejected.

(2) There are not many rejected points[3].
(3) The function restricted to the set of accepted points is monotone.

The third property ensures that it is possible (though not necessarily efficiently) to change the function only on rejected points and make the function monotone. To do this, define $m(x)$ for $x \in \Gamma$ to be the largest accepted point less than or equal to x, and define $g(x) = f(m(x))$. It is easy to see that this yields a monotone function.

In [3], a point x is rejected if (roughly) there is an interval around x that contains a large fraction of points whose f values are out of order with respect to $f(x)$. With this accepted/rejected classification there seems to be no fast way to compute $m(x)$. Instead, given a query point x, the filter in [3] selects a sample of points less than or equal to x (called the *sample* of x), chooses $m'(x)$ to be the largest accepted point in the sample, and defines $g(x) = g(m'(x))$. The sampling procedure chooses $z < x$ with probability (roughly) inversely proportional to the distance of z from x; in particular the sample includes x itself, so if x is accepted then $m'(x) = x$.

Defining g in this way creates a problem: g need not be monotone. For example, let y be a point and $x = m(y) < y$ be the largest accepted point less than or equal to y. Suppose that a query is made to y and x is not in the sample of y, so $m'(y) < x$. Suppose further that $f(m'(y)) < f(x)$. Suppose that after setting $g(y)$ to $f(m'(y))$, a query is made to index x. Since x is an accepted point we will have $m'(x) = x$ and so $g(x) = f(x)$, but this will violate monotonicity with the already defined $g(y) = f(m'(y))$.

In online reconstruction, this is not a significant problem because the algorithm can save the previously answered queries in a sorted list and impose the condition that future g values be consistent with previously assigned g values. This is what is done in [3].

Local reconstruction does not have this luxury. What we do is to redefine $m'(x)$ so as to guarantee that for any $y > x$, we have $m'(y) \geq m'(x)$. To do this, after sampling the points less than x we identify certain points of the sample which have the potential for creating non-monotonicities and exclude them from the sample. For example, in the scenario above, the point x needs to be excluded from its own sample to avoid the *potential* non-monotonicity with y. Notice that when we exclude x from its own sample we may introduce a new point where $g(x) \neq f(x)$, so we cannot do this too often.

Thus, the main challenge in designing a local filter is to find an efficient way to identify the points that need to be excluded from the sample of x to avoid potential non-monotonicities. We also need to ensure that x is not excluded from its own sample too often.

The difficulties in designing a local filter are greater for the case of higher-dimensional domains ($d \geq 2$). Suppose we had a definition of accepted and rejected satisfying the three conditions stated in the one-dimensional case. In principle, it is still possible to define a monotone g that agrees with f on all accepted points. But explicitly computing such a g is more complicated. Given x, let $M(x)$ be the set of points which are maximal in the set of accepted points less than or equal to x. In the one-dimensional case, $M(x)$ has one element $m(x)$, but in the multi-dimensional case, where the domain is not totally ordered, this is not the case. Still, if we define $g(x)$ to be the maximum of $f(y)$ for $y \in M(x)$, then the resulting g is monotone. To implement this, one would have to find all of the elements of $M(x)$. Even when $M(x)$ has size 1 (as in the one-dimensional case) this is difficult, but here the difficulty is compounded because $M(x)$ might be as large as $\Omega(n^{d-1})$, and we need our computation to run in time polylogarithmic in n. In [3], this is handled by finding a polylogarithmic size sample that is a suitable approximation to $M(x)$, and then defining $g(x)$ to be the maximum of $f(y)$ for y in the sample.

As with the one-dimensional case, using an approximation to $M(x)$ destroys the guarantee that g

[3]The number of rejected points is comparable to the distance of f to monotonicity.

defined in this way is monotone. Hence, one must save the values of g to all queries, and impose the additional requirement that queries are mutually consistent. Since a local filter cannot save these values, we again need to judiciously exclude points from the sample to avoid non-monotonicities.

The definition of the sample, which we denote $\textsc{Rep}(x)$, crucially uses a data structure of nested boxes (products of intervals). Condition (3) is maintained by a careful and efficient scheme for passing crucial information about the distribution of rejected points in a particular box to its sub-boxes.

References

[1] N. Ailon and B. Chazelle. Information theory in property testing and monotonicity testing in higher dimension. In *Proceedings of the 22nd Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 434–447, 2005.

[2] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Estimating the distance to a monotone function. In *Proceedings of the 8th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 229–236, 2004.

[3] N. Ailon, B. Chazelle, S. Comandur, and D. Liu. Property preserving data reconstruction. In *Proceedings of the 15th Annual International Symposium on Algorithms and Computation (ISAAC)*, pages 16–27, 2004.

[4] N. Alon, E. Fischer, I. Newman, and A. Shapira. A combinatorial characterization of the testable graph properties : it's all about regularity. In *Proceedings of the 38th Annual Symposium on Theory of Computing (STOC)*, pages 251–260, 2006.

[5] N. Alon and A. Shapira. A charaterization of the (natural) graph properties testable with one-sided error. In *Proceedings of the 46th Foundations of Computer Science (FOCS)*, pages 429–438, 2005.

[6] S. Arora and M. Sudan. Improved low-degree testing and its applications. In *Proceedings of the 29th Annual Symposium on Theory of Computing (STOC)*, pages 485–495, 1997.

[7] T. Austin and T. Tao. On the testability and repair of hereditary hypergraph properties. Technical Report arXiv:0801.2179v2, May 2009.

[8] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking computations in polylogarithmic time. In *Proceedings of the 23rd Annual Symposium on Theory of Computing (STOC)*, pages 21–31, 1991.

[9] L. Babai, L. Fortnow, N. Nisan, and A. Wigderson. PP has subexponential time simulations unless EXP-TIME has publishable proofs. *Computational Complexity*, 3:307–318, 1993.

[10] T. Batu, R. Rubinfeld, and P. White. Fast approximate PCPs for multidimensional bin-packing problems. *Information and Computation*, 196(1):42–56, 2005.

[11] A. Bhattacharyya, E. Grigorescu, K. Jung, S. Raskhodnikova, and D. Woodruff. Transitive-closure spanners. In *Proceedings of the 18th Annual Symposium on Discrete Algorithms (SODA)*, pages 531–540, 2009.

[12] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993.

[13] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. *Journal of the ACM*, 45:965–981, 1998.

[14] Y. Dodis, O. Goldreich, E. Lehman, S. Raskhodnikova, D. Ron, and A. Samorodnitsky. Improved testing algorithms for monotonicity. *Proceedings of the 3rd International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*, pages 97–108, 1999.

[15] F. Ergun, S. Kannan, R. Kumar, R. Rubinfeld, and M. Viswanathan. Spot-checkers. *Journal of Computer Systems and Sciences (JCSS)*, 60(3):717–751, 2000.

[16] S. Fattal and D. Ron. Approximating the distance to monotonicity in high dimensions. In *http://www.eng.tau.ac.il/ danar/Public-pdf/app-mon-long.pdf.*

[17] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bulletin of EATCS*, 75:97–126, 2001.

[18] E. Fischer. On the strength of comparisons in property testing. *Information and Computation*, 189(1):107–116, 2004.

[19] E. Fischer, E. Lehman, I. Newman, S. Raskhodnikova, R. Rubinfeld, and A. Samorodnitsky. Monotonicity testing over general poset domains. In *Proceedings of the 34th Annual Symposium on Theory of Computing (STOC)*, pages 474–483, 2002.

[20] P. Gemmell, R. Lipton, R. Rubinfeld, M. Sudan, and A. Wigderson. Self-testing/correcting for polynomials and for approximate functions. In *Proceedings of the 23rd Annual Symposium on Theory of Computing (STOC)*, pages 32–42, 1991.

[21] O. Goldreich. Combinatorial property testing - a survey. *Randomization Methods in Algorithm Design*, pages 45–60, 1998.

[22] O. Goldreich, S. Goldwasser, E. Lehman, D. Ron, and A. Samordinsky. Testing monotonicity. *Combinatorica*, 20:301–337, 2000.

[23] O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

[24] S. Halevy and E. Kushilevitz. Testing monotonicity over graph products. *Random Structures and Algorithms*, 33(1):44–67, 2008.

[25] J. Katz and L. Trevisan. On the efficiency of local decoding procedures for error-correcting codes. In *Proceedings of the 32th Annual Symposium on Theory of Computing (STOC)*, pages 80–86, 2000.

[26] M. Parnas, D. Ron, and R. Rubinfeld. Tolerant property testing and distance approximation. *Journal of Computer and System Sciences*, 6(72):1012–1042, 2006.

[27] D. Ron. Property testing. *Handbook on Randomization*, II:597–649, 2001.

[28] R. Rubinfeld and M. Sudan. Robust characterization of polynomials with applications to program testing. *SIAM Journal of Computing*, 25:647–668, 1996.

[29] M. Saks and C. Seshadhri. Parallel monotonicity reconstruction. In *Proceedings of 19th Annual Symposium on Discrete Algorithms (SODA)*, pages 962–971, 2006.

[30] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the *XOR* lemma. *Journal of Computer and System Sciences*, 62(2):236–266, 2001.