JOURNAL OF ALGORITHMS 19, 161–172 (1995)

Fractional Cascading Revisited*

Sandeep Sen

Department of Computer Science and Engineering, Indian Institute of Technology, Delhi, New Dehi 110016, India

Received December 3, 1990; revised March 1994

We present an alternative implementation of the fractional cascading data structure of Chazelle and Guibas (*Algorithmica* 1 (1986), 133–162) that performs iterative search for a key in multiple ordered lists. The construction of our data structure uses randomization and simplifies the algorithm of Chazelle and Guibas, vastly making it practical to implement. Although our bounds are asymptotically similar to the earlier ones, there are improvements in the constant factors. Our analysis is novel and captures some of the inherent difficulties associated with the fractional cascading data structure. In particular, we use tools from branching process theory and derive some useful asymptotic bounds. The probability of deviation from the expected performance bounds decreases exponentially with number of keys. Also, under a restricted model, we obtain efficient bounds for updates in the data structure.

1. INTRODUCTION

The problem of searching for a key in many ordered lists arises very frequently in computational geometry (see Chazelle and Guibas [2] for applications.) Chazelle and Guibas [1] introduced fractional cascading as a general technique for solving this problem. Their work unified some earlier work in this area and gave a general strategy for improving upon the naive method of doing independent searches for the same key in separate lists. In brief, they devised a data structure that would enable searching for the same key in n lists in time $O(\log M + n)$ where M is the size of the longest list. If N is the total size of all the lists then this data structure can be built in $O(N)$ preprocessing time and takes $O(N)$ space. We shall give a more precise description of their data structure in the next

* Part of this research was done when the author was a post-doctoral Member of Technical Staff at AT & T Bell Labs, Murray Hill, New Jersey, during the year 1990–1991. A preliminary version of this paper appeared in the "Proceedings of the 3rd Scandinavian Workshop on Algorithmic Theory, Helsinki, Finland, July 1992."

0196-6774/95 $12.00

section. Their solution, although, elegant, is difficult to implement, and they leave open the question of simplifying it to be useful in practice.

In this paper, we give an alternative implementation of their data structure that uses randomization. While retaining the salient features of their data structure, we are able to simplify its construction considerably to an extent that is practical. The motivation of the new technique has been derived from skip lists (Pugh [10]). However, our method requires new analytical techniques that may have further applications. In particular, we use tools from branching-process theory and derive some useful asymptotic bounds. It may be noted that this technique (now known as dynamic sampling) had its origin in an earlier version of the present work [12] and Sen [13]. Subsequently these ideas have been generalized in the context of dynamic maintenance of data structures by Mulmuley and Sen [9] and Mulmuley [6, 7].

Our work still leaves open the issue of dynamic maintenance of fractional-cascading data structure that attains optimal performance. However, it does simplify part of the scheme which turns out to be useful for obtaining optimal bounds for *random updates*. We describe this mode in Section 4. The bounds for space and preprocessing time that we obtain for the static case are expected worst case and hold with probability $1 - 2^{-\Omega(N)}$. The search time holds with high probability, i.e., $1 - 1/N^k$ for any fixed $k > 0$. The following notation will be used in the paper. We say a randomized algorithm has resource (time, space, etc.) bound $\tilde{O}(g(n))$ if there is a constant c such that the amount of resource used by the algorithm (on any input of size n) is no more than $c\alpha g(n)$ with probability $\geq 1 - 1/n^{\alpha}$, for any $\alpha \geq 1$.

2. DESCRIPTION OF FRACTIONAL CASCADING

In this section, we give a brief description of the problem setting and the approach taken by Chazelle and Guibas. Consider a fixed graph $G = (V, E)$ of $|V| = n$ vertices and $|E| = m$ edges. The graph G is undirected and connected and does not contain multiple edges. Each vertex v has a catalog C_v and associated with each edge e of G is a range R_e.

A catalog is an ordered collection of records where each record has an associated value in the set $\Re \cup \{\infty, -\infty\}$. Note that the value of a record is distinct from the record. The records are stored in a nondecreasing order by their values, and more than one record can have the same value. A catalog is never empty (has at least a ∞ and a $-\infty$). A range is an interval of the form $[x, y], [-\infty, y], [x, \infty]$, or $[-\infty, \infty]$. The graph G together with the associated catalogs and ranges is called a catalog graph. This is the basic structure to which fractional cascading is applied.

For notational purpose, if the value k is an endpoint of the range $R_{u,v}$, then k appears as the value of some record in both C_u and C_v. Moreover, if two ranges $R_{u,v}$ and $R_{v,w}$ have an endpoint in common, it appears twice in C_v. The space required to store a catalog graph in $N = \Sigma_{v \in V} |C_v|$. This includes the space to store the graph itself. A catalog graph G is said to be *locally bounded* by degree d if for each vertex v and each value of $x \in \Re$ the number of edges incident on v whose range includes x is bounded by d.

The input to a query is an arbitrary element k in the universe and a connected subtree $\Pi = (\bar{V}, \bar{E})$ such that $k \in R_e$ for all edges e in the subtree and $\bar{V} \subset V$, $\bar{E} \subset E$. The output of the query for each vertex $v \in \bar{V}$ is an element y such that $predecessor(y) < k \leq y$. We shall refer to the pair $(predecessor(y), y)$ of elements as the *straddling pair* of k.

THEOREM 1 (CHAZELLE-GUIBAS). *Let G be a catalog graph of size N and locally bounded degree d. In $O(N)$ space and $O(dN)$ time it is possible to construct a data structure that allows multiple look-ups (query) in a subtree of size p in time $O(pd + \log N)$. If d is sized this is optimal. In addition, if the underlying catalog graph G is restructured to a constant degree graph, then the search time and the preprocessing time can be improved to $O(p \log d + \log N)$ and $O(N)$, respectively.*

3. ANATOMY OF THE DATA STRUCTURE

Our data structure is very similar to that of [1] in the sense that we retain their idea of using an augmented catalogs A_v for every vertex v such that $C_v \subset A_v$. But we shall use a different method for the construction of the augmented catalogs. An augmented catalog A_v is also a linear list of records whose values form a sorted multiset. Augmented catalogs in neighboring nodes of G will contain a number of records with common values. The corresponding records are linked together to correlate locations in the two catalogs. The objective is that given the location of a record in A_v, we would be able to find its location (the straddling pair) in the augmented catalog of a neighbor of v in constant additional time. More formally, for each node u and an edge e connecting u and v in G, we maintains a list of "bridges" from u to v, B_{uv}, which is an ordered subset of records in A_v and that are contained in the range R_e. The endpoints of R_e are the first and last records of B_{uv}. In node v, we maintain for each bridge in B_{uv} a companion bridge in B_{vu}. Recall that the value of a record is distinct from the record. Moreover each bridge is associated with a unique edge of G, implying that if a given value in A_u is chosen to be a bridge in both B_{uv} and B_{uw}, then it is duplicated and stored in different records of A_u.

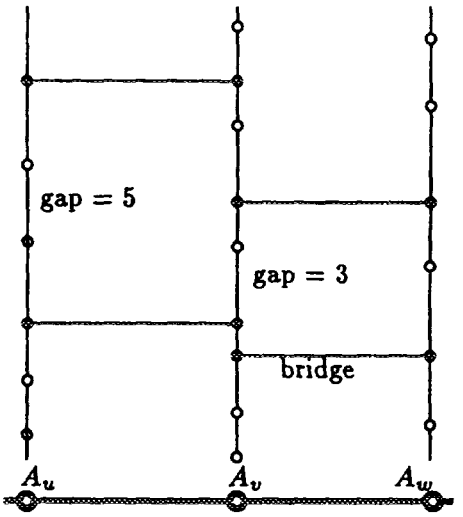

FIG. 1. Gaps, bridges, and augmented catalogs.

A pair of consecutive bridges associated with the same edge $e = (u, v)$ defines a *gap*. Let a_u and b_u be two consecutive bridges in B_{uv} and a_v (respectively, b_v) be the companion bridges in B_{vu}. If $value(a_u) < value(b_u)$, then the gap of b_u includes all elements of A_u positioned strictly between a_u and b_u and all elements of A_v between a_v and b_v (the bridges are not included). By definition, the gap of b_v is the same as gap of b_u. One of the key strategies used by Chazelle and Guibas [1] is to maintain the invariant that no gap exceeds $6d - 1$ in size.

We now take a closer look at the information maintained with each record. Both C_v and A_v are maintained as linked lists. A record of C_v has the fields *key* and *up-pointer*. The key contains the value and the up-pointer is a pointer to the next record. Each record of A_v has several other fields:

(1) key: stores the value k of record r.

(2) C-pointer: holds a pointer $\nu(r)$, the successor of r in C_v.

(3) up-pointer, down-pointer: pointers to successor and predecessor in A_v.

In addition, a bridge element also has pointers to its companion bridge and the label of the edge for which it is a bridge. If r is a bridge in B_{uv} then it stores the label uv.

To answer a multiple look-up query (x, Π), where x is the key value and Π is the subtree, one begins by locating x in the first node of the subtree Π and then uses the following properties:

LEMMA 1 (CG). *If we know the position of value x in A_v, we can compute the position of x in C_v in one step.*

This can be done by using the C-field pointer.

LEMMA 2. *If we know the position of value x in A_v and $e = (v, w)$ is an edge of G such that $x \in R_e$, then we can compute the position of x in A_w in $O(|Gap_e(x)|)$ time. $Gap_e(x)$ is the set of elements in the gap (corresponding to edge e) which x points at.*

From the position of x in A_v follow up-pointers until a bridge is found that connects to A_w. Note that Chazelle and Guibas [1] maintained the invariant that all gap sizes are less than $6d$, which yields a search time of $O(\log N + d|\Pi|)$. This invariant was maintained during the construction of all the augmented catalogs which is done incrementally. Their algorithm starts with empty catalog and then for each vertex v, the records of C_v are inserted in increasing order into A_v. Between any two insertions, the gap invariants are restored. Note that a single insertion into A_v can alter the gaps leading to insertion of a new bridge which introduces new records and this could continue as a long chain of events.

We propose the following modification. Instead of explicitly maintaining gap invariants, we choose a newly inserted element r in A_v to be a bridge with probability p. (We shall determine the exact value later.) This process is repeated for each edge incident on v whose range covers r. If r is chosen to be a bridge for an edge (v, w), it leads to the insertion of a new record in A_w and possibly even in A_v if x is already a bridge. These new records are treated exactly the same way as described above (i.e., choose each with probability p to be a bridge element). For each new record in the augmented catalog, we initialize the following fields:

(i) C-pointer: Can be determined from the predecessor and the successor elements in the augmented catalog. If this element came from C_v, then update the C-pointers for all the elements in A_v between this record and the previous record of C_v.

(ii) Edge field: if the element is a bridge element, then it contains the edge for which it is a bridge.

For each new record of C_v, this process is continued until there are no more bridges to insert.

4. ANALYSIS

Let us first analyze the running time for a multiple look-up query. Given the position of x in A_v we follow the up-pointers until we find a bridge b_v that connects to A_u. Since each element is chosen to be a bridge element with probability p, the expected length of a gap is $2/p$. Thus from Lemma 1, the expected search time is $O(\log N + |\Pi|/p)$.

Moreover, if $|\Pi| \geq \log N$, we can show that the search time is $\tilde{O}(\log N + |\Pi|/p)$ using the observation that the search time is a sum of $O(\log N)$ independent random variables with a geometric distribution and parameter p (Sen [11]). There is however an oversight in the above reasoning since the query is a tree of size $|\Pi|$ which can assume various forms. In particular, Π can be any of the positional trees of maximum degree d. A positional tree has its edges out of any vertex labeled from the set $\{0, 1, \ldots, d-1\}$ where d is the maximum degree. It can be shown that there are $2^{\Omega(\log d|\Pi|)}$ such trees for $\Pi < \sqrt{n}$. When $1/p$ is proportional to d (as we shall prove later) and d is bounded, then the search time holds with high likelihood. Note that there are $O(N)$ choices for the root of this tree and $O(N)$ combinatorially distinct search paths given the search tree (corresponding to the $O(N)$ intervals induced by all the key values).

LEMMA 3. *The new scheme allows multiple query in a subtree Π in expected time $O(\log N + |\Pi| \cdot d)$. If the local degree d is bounded by a constant then the query time is $\tilde{O}(\log N + |\Pi|)$; i.e., the bound holds with high probability.*

Remark. If Π is $\Omega(n)$, then the number of positional trees is bounded by c^n for some constant c (see Theorem 10 in Manber and Tompa [5]). Hence the search time exceeds $O(nd)$ with probability less than $2^{\Omega(n)}$.

The more challenging aspect of the analysis is to bound the time and space required to construct the data structure. If we look more closely at the way the records are added to the augmented catalogs, we can model the underlying stochastic process by a branching process. The root corresponds to an inserted record from C_v, and the children correspond to the bridges that are created by this record. Each bridge that corresponds to two children, is created with probability p. For each new record inserted from the C_v's the time and the space needed is proportional to the total progeny of this branching process. Each node can have up to $2d$ children where the actual number of children is a random variable that takes values $0, 2, 4, \ldots 2d$. The probability that this random variable takes value $2k$ is the same as the probability that a binomial random variable with parameters (d, p) takes value k (i.e., there are k bridges created). The mean μ of

this distribution is clearly $2pd$ and the generating function $G(s)$ can be derived to be $(q + ps^2)^d$. Here $q = 1 - p$.

From Feller [3], a branching process is finite if $\mu < 1$. Hence we choose p such that $2pd < 1$, that is, $p < 1/2d$. This gives an expected gap length of greater than $4d$. From [3], if the generating function for the total progeny is denoted by $t(s)$, then $t = sG(t)$. In our case $G(t) = (q + pt^2)^d$. Moreover, the mean is $1/(1 - \mu)$, which is $1/(1 - 2pd)$ in our case. If we choose $p = 1/3d$, this yields a mean progeny of 3. This in turn implies a total expected space bound of $O(\Sigma_{v \in V} C_v)$ which is $O(N)$.

We can get stronger bounds by estimating the probability of deviating from the mean value. The usual procedure is to use Chernoff bounds but in our case it is complicated by the fact that we cannot get an explicit generating function for $d > 2$ (since it involves solving equations of high degree). Instead we take an indirect approach. The total space and time for construction of the data structure is the sum of N independent and identical random variables Y_i, $1 \le i \le N$, each of which is the total progeny of a branching process.

If $A = \Sigma_i Y_i$, then from Chernoff bounds,

$$Prob[A \ge X] \le s^{-X} t(s)^N, \tag{1}$$

where $t(s)$ is the generating function for each Y_i. For $X > cN$, for some fixed c, this can be rewritten as

$$Prob[A \ge cN] \le \left(\frac{t(s)}{s^c}\right)^N.$$

We shall prove that for some $s > 1$, there exists a constant c such that $t(s)/s^c < 1$. Let $F(s,t) = t - s(q + pt^2)^d$. Then

$$F_t(s,t) = 1 - 2tpds(q + pt^2)^{d-1}$$

Hence $F_t(1,1) = 1 - 2pd$ and for $2pd < 1$, $F_t(1,1) \neq 0$. From the *Implicit Function Theorem* (see Appendix) it follows that there exists a neighborhood of $(s = 1, t = 1)$, such that

$$F(s,t) = F(1,1) \Leftrightarrow t = t(s).$$

Since $F(1,1) = 0$ this implies that there is a value $s > 1 + \epsilon$ for which $t(s) < 1 + \delta$ for some $\epsilon, \delta > 0$ independent of N. By choosing c large enough, $t(s)/s^c$ can be made less than 1 and hence the probability of deviation from the mean decreases as $1/2^{\Omega(N)}$.

For each new record in A_v, we need $O(d)$ time to determine if it will be a bridge with respect to any of the d (maximum) neighbors. Moreover

inserting a new bridge takes time proportional to gap size whose expected value is $O(d)$. Thus the total time for inserting $O(N)$ bridges is a sum of $O(N)$ independent geometric random variables with parameter p. (Unlike the argument for ensuring search time with high probability, where we needed to consider all distinct query trees, we simply bound this sum.) This is $O(Nd)$ with probability $1 - 2^{-\Omega(N)}$ using standard techniques like Chernoff bounds.

To complete the analysis for the time bound for building the data structure, we have to ensure that the total number of C-pointer updates is also $O(N)$. For any new record inserted into A_v from C_v, the total number of C-pointer updates can be bound by the total number of records (i.e., the space bound). The records of C_v are inserted in an increasing order and thus any record in A_v has its C-pointer updated at most once (not including when it is first created). Hence we state the following result

THEOREM 2. *Let G be a catalog graph of size N and locally bounded degree d. Our algorithm constructs a data structure for iterative search in $O(N)$ space and $O(dN)$ time that does iterative search in expected time $O(\log N + d|\Pi|)$. The bounds for preprocessing time and space hold with probability $1 - 2^{-\Omega(N)}$.*

Remarks.

(1) The bounds for preprocessing time and search time can be improved to $O(N)$ and $\tilde{O}(\log N + \log d|\Pi|)$, respectively, by using the same modifications as Chazelle–Guibas to restructure the catalog graph to a fixed degree graph.

(2) Chazelle and Guibas also arrived at the figure $4d$ for minimum gap size from the observation that a lower gap size in their analysis leads to infinite time and space bound for construction. However, they give examples where the actual algorithm halts even when they use gap sizes of less than $4d$. Our analysis captures a more fundamental reason for this phenomenon. Although the mean $\mu < 1$ guarantees that the process is finite, the process dies out with probability x where $x = G(x)$ even when $\mu > 1$. Thus one can have a gap length of less than $4d$ and still terminate. The motivation for this is clearly a reduction in search time which is inversely proportional to the gap size.

(3) To allow insertions/deletions from the catalogs, our procedure for maintaining the augmented catalogs readily dynamized. The arguments for query time and the space bound remain identical to those in the static case. Unfortunately (as in the case of [1]), the bottleneck is maintaining

the correspondence between the A_v and C_v. In particular, we are unable to get a meaningful bound on the number of C-pointer updates in the case of inserting or deleting a record from C_v. If the priority queue of Fries *et al.* [4] is used to maintain this correspondence, both the search time and update times are off by an $O(\log \log N)$ factor from the best possible.

Implementation of the two schemes was carried out on a SUN workstation for a comparative study. For data, random graphs of about 100 nodes were generated, where each node had catalogs of about 1000 records. Each record of the augmented catalog in the randomized case required 34 bytes as compared to 48 of the deterministic scheme. The number of bridges created in the deterministic case was twice that of the randomized scheme. This implies saving of a factor of 2.5 in space requirements. The mean progeny appeared closer to $1/(1 - 2pd_{av})$ than the predicted $1/(1 - 2pd)$ where d_{av} was the average degree. Moreover the number of bridges added due to a single insertion was always more in the deterministic case. The number of pointer traversals was also more than a factor of two greater in the deterministic case. Experiments confirmed that for $p > 1/2d$ (gap size smaller than $4d$), the construction terminated.

An interesting feature of the randomized scheme was the ease with which it could be made dynamic, i.e., one where insertions and deletions of records are allowed. Note that the gap sizes do not have to be maintained explicitly; thus there is no modification in the basic procedure. The procedure for deletion is exactly the inverse of insertion. While worst-case scenarios can be constructed which requires changing a large number of C-pointers, the experiments showed that it has excellent behavior for the data used. This could be attributed to the following reasons. Unlike the deterministic case where the *amortized* insertion time is bound, the insertion time in the randomized scheme is (expected) *worst* case. Moreover, for *random* insertions (and deletions), the expected number of C-pointer updates for any fixed record is actually constant.

By random updates, we imply that the records in the catalogs are inserted (deleted) in a random order where all permutations are equally likely. The N record values themselves can be chosen arbitrarily. This model was used by Mulmuley [6] to obtain efficient bounds for a number of dynamic schemes in computational geometry. It may be useful to view this model as the following—at any instance when r objects are present, all subsets of size r among the universe of N objects are equally likely to occur. If the next operation is an insertion then any of the remaining objects is equally likely to be inserted. If the next operation is deletion then any of the objects present is equally likely to be deleted. The number of C-pointer updates of a fixed record R in an augmented catalog A_v is

the number of records in the catalog C_v *adjacent* (successors in the catalog) to R over a sequence of $O(n)$ updates.

We can use *backward analysis* to get a bound on the expected number of C-pointer changes for a sequence of $O(n)$ random updates involving n records of a C_v. Let W_j be 1 if the record adjacent to R changes at the jth update step, i.e., causes the C-pointer of R to change. The probability that W_j is 1 is then the probability of the event that R is present in A_v and one of the adjacent records in C_v changes. Note that for the purpose of this analysis, we can ignore the remaining records of A_v and look at the relative orderings of R and only the n records of C_v. Moreover, we can pretend that all the records in the A_v's are known to us because the bridge elements are generated independently of the other records. Let r be the number of records currently in C_v including R. From backward-analysis argument, the probability that an inserted record is adjacent to R is $1/(r+1)$ by conditioning on the set of size $r+1$ (including the inserted element) and deleting a random record. The probability that a deleted record is adjacent to R is $1/r$ from the previous paragraph. Thus the probability that W_j is 1 can be bounded by $r/n \cdot 1/r \leq 1/n$. The expected value of W_j is $O(1/n)$ and the expected value of $\sum_j W_j$ is $O(1)$. The expected number of C-pointer changes for any fixed record caused by $O(N)$ updates over all the catalogs can be bounded by $O(1)$. The updates in the catalogs where R is not present do not matter. Since the total number of records is $O(N)$ (space bound), and the mean progeny is $O(1)$ for each record we obtain the following result.

THEOREM 3. *For a sequence of $O(N)$ random updates in the catalogs involving a total of N records, the expected amortized cost for an update is $O(\log N + d)$, including the time for searching.*

5. CONCLUDING REMARKS

Two of the outstanding problems posed by Chazelle and Guibas [1] still remain open, namely, the issue of high local-degree graphs and maintaining same asymptotic bounds in the dynamic scenario under worst-case update. Both in the Chazelle and Guibas scheme and in our scheme, the gap size seems to be tied inherently with the local degree of the graph. In this respect, our analysis seems to provide a more concrete explanation, namely its connection with the convergence of a branching process. We conjecture that the gap size can be made proportional to the local degree at each node instead of the maximum local degree of the graph—analyzing such a branching process however appears to be very complicated.

For the dynamic part, our experiments suggest that our algorithm has very good average performance even without the schemes of Fries *et al.* which were used by Chazelle and Guibas along with other complicated data structures. We may mention here that in the context of dynamic maintenance of data structures two distinct approaches have been proposed: the dynamic sampling method [6, 7, 9] and the randomized multidimensional search trees [8]. It is unclear how the latter methodology can be adapted for our case since the bridges of the augmented catalog have a natural correspondence to a sample and the only structures that we maintain are linked lists.

ACKNOWLEDGMENTS

The author is very grateful to Paul Wright who pointed out the use of the Implicit Function Theorem, to show existence of the generating function for $s > 1$. The author also acknowledges Praveen Singh who implemented the two schemes as part of his Bachelor's thesis.

APPENDIX: IMPLICIT FUNCTION THEOREM

We use the standard notation f_y to denote the partial derivative of f with respect to variable y.

THEOREM 4. *Let $f(x, y)$ be continuously differentiable in D. Let (x_0, y_0) be any point in D such that $f_y(x_0, y_0) \neq 0$. Then there exist numbers $\delta > 0$ and $\epsilon > 0$ and a continuously differentiable function $g(x)$ defined for $|x - x_0| \leq \delta$ and $|y - y_0| \leq \epsilon$, then $f(x, y) = f(x_0, y_0) \Leftrightarrow y = g(x)$.*

REFERENCES

1. B. Chazelle and L. Guibas, Fractional cascading: I. A data structuring technique, *Algorithmica* **1** (1986), 133–162.
2. B. Chazelle and L. Guibas, Fractional cascading: II. Applications, *Algorithmica* **1** (1986), 163–191.
3. W. Feller, "An Introduction to Probability Theory and Applications Vol. I," Wiley, New York, 1968.
4. O. Fries, K. Mehlhorn, and S. Näher, Dynamization of geometric data structures, *in* "Proceedings, 1st ACM Computational Geometry Symposium, 1985," pp. 168–176.
5. U. Manber and M. Tompa, The effect of number of Hamiltonian paths on the complexity of a vertex-coloring problem, *SIAM J. Comput.* **13**(Feb.) (1984), 109–115.
6. K. Mulmuley, Randomized multidimensional search trees: Dynamic sampling, *in* "Proceedings, 7th ACM Symposium on Computational Geometry, 1991," pp. 121–131.
7. K. Mulmuley, Randomized multidimensional search trees; Further results in dynamic sampling, *in* "Proceedings, 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 216–227.

8. K. Mulmuley, Randomized multidimensional search trees: Lazy balancing and dynamic shuffling, *in* "Proceedings, 32nd IEEE Symposium on Foundations of Computer Science, 1991, pp. 180–196.
9. K. Mulmuley and S. Sen, Dynamic point location in arrangements of hyperplanes, *Discrete Comput. Geom.* **8** (1992), 335–360.
10. W. Pugh, Skip lists: A probabilistic alternative to balanced trees, *Commun. ACM*, **33**(6) (1990), 668–676.
11. S. Sen, Some observations on skip lists, *Inform. Process. Lett.*, **39** (1991), 173–176.
12. S. Sen, Fractional cascading revisited, Bell Labs technical memorandum, Murray Hill, NJ, 1991.
13. S. Sen, Maintaining arrangements for point location, Bell Labs technical memorandum, Murray Hill, NJ, 1991.