(*Extended Abstract*)

Yet Another Look at Fractional Cascading: B-graphs with Application to Point Location*

Chee Yap and Yunyue Zhu
Department of Computer Science
Courant Institute, New York University

July 17, 2001

Abstract

We revisit the notion of catalog graphs which was formulated by Chazelle and Guibas as a framework for fractional cascading. We introduce the **catalog DAG** which incorporates several modifications to make the model more realistic and implementable. Simultaneously, we propose the **B-graph data structure** as a generalization of the B-trees of Bayer and McCreight. This turns out to be a natural structure for implementing fractional cascading in our reformulated catalog DAGs. The practical implementation of fractional cascading has been a subject of research, and B-graphs offer a solution. We apply B-graphs to the problem of point location, extending Seidel's randomized incremental algorithm to monotone chains. If the n input segments are partitioned into N monotone chains ($1 \le N \le n$), then the expected number of B-nodes that our algorithm visits is $\log_b(n/N) + 4H_N$ where b is the parameter of B-trees and H_N is the N-th harmonic number. Thus we obtain an I/O-efficient algorithm. Taking $b = 2$, this also yields a very favorable constant compared to Seidel's algorithm. Experimental results on cartographic data (specifically, the TIGER dataset) will be described.

1 Introduction

Motivated by a growing list of examples in computational geometry in the early 1980s, Chazelle and Guibas [3, 4] introduced **fractional cascading** as an efficient strategy for solving the **multiple look-up query problems**. Their formulation is difficult to implement, and they left open the question of simplifying it to be practical. Sen [13] simplified fractional cascading using a skip-list type randomization. In this paper, we propose another approach based on B-trees. It should be noted that most of the papers (e.g., [3, 10]) in this area were strongly motivated by the B-trees of Bayer and McCreight [2]. The fractional cascading data structure is often described as a generalization of B-trees. In [10], the parameters of (a, b) from B-trees are used in designing fractional cascading. Still, all previous identifications with B-trees fail in many aspects. We will introduce the B-graph data structure to close this gap. B-trees have proven to be one of the most useful search structures in practice. Two advantages of B-trees, which our B-graphs also accrue, are their relatively simple update (insert/delete) algorithms and the ability to achieve good secondary memory (I/O) performance. Agarwal et al [1] have already exploited B-trees to achieve I/O-efficient algorithms for this problem. But their result does not exploit the presence of monotone chains, which can give considerable practical speedups as well as improve space efficiency.

One application of fractional cascading is to the problem of point location for monotone chains [6]. We consider point location in cartography, specifically for the TIGER dataset [14] from the US Census Bureau. The TIGER subdivision of the plane is defined by polygonal chains (called TIGER lines). If we reduce point location to a planar triangulation (as in Kirkpatrick's algorithm [8]) or to a trapezoidal map (as in Seidel algorithm [12]), the size of TIGER dataset will blow up by a small but non-neglible constant factor. This is unrealistic for large datasets. Instead, we subdivide TIGER lines into monotone chains, and considerably reduce this overhead. Since Seidel's algorithm (which is in turn de-

*This paper will be presented at the 13th Canadian Conference on Computational Geometry, University of Waterloo, Aug 13-15, 2001. The work is partially supported by NSF/ITR Grant #CCR-0082056.

rived from Mulmuley's work) is considered one of the most practical algorithm in this area, we generalize this algorithm to monotone chains. Seidel's, Kirkpatrick's and several other $O(\log n)$ time algorithms are already asymptotically optimal as a function of n, the total number of line segments. But there is practical interest in reducing the implicit constants in these algorithms. If we count the number of comparisons, Seidel's algorithm has an expected bound of $12H_n$ [5, p.135] (the constant is worse in Kirkpatrick's algorithm). Now suppose the n segments are further partitioned into N monotone chains ($1 \leq N \leq n$). We generalize Seidel's algorithm to perform a random insertion of these chains to create a generalized trapezoidal map. A similar analysis shows that the expected depth of our chain structure is $4H_N$. Other effects such as fractional cascading and the B-tree parameter b comes into play in our data structure, and we finally obtain:

Theorem 1 *The expected number of B-nodes visited by any point location query in the above search structure is* $T = \log_b(n/N) + 4H_N + O(1)$.

This expresses the I/O complexity of our algorithm. To compare this result to Seidel's "main-memory" bound, we may set the B-tree parameter to $b = 2$. In this case, $T \leq 4 \lg n$, a considerable improvement over Seidel's constant of 12. We implemented Seidel's original algorithm as well as a fractional cascading version of our B-graph algorithm. Experimental results will be reported in the full paper.

Contributions of this paper.

- We reexamine the formulation of catalog graphs of Chazelle and Guibas. In particular, its notion of "bridges" has been adopted by all subsequent authors in fractional cascading. This turns out to be a major impediment to the implementation of fractional cascading. We propose a relaxed version called "semibridges" which is sufficient for fractional cascading. This notion is easier to maintain and is more space efficient.

- Our reformulation of catalog graphs (called catalog DAGs) introduces the natural notions of edge types and discriminant functions. They allow us to directly model the query formation process which falls outside the original catalog graph model.

- We introduce B-graphs as the data structure counterpart of catalog DAGs. This is a now a true generalization of B-trees, and inherits the well-known advantages of B-trees for I/O efficient algorithms.

- We generalize Seidel's point location algorithm for line segments to monotone chains, and apply it to the well-known cartographic TIGER dataset which is based on chains.

- Ongoing experimental work hopes to confirm our belief that our new approach could yield one of the most practical point location algorithms to date. We expect to experimentally show the improvements in the asymptotic time/space constants over Seidel's algorithm, to accrue the I/O efficiency of B-trees, and to exploit the presence of monotone chains in our point location subdivision. This will appear in the final paper.

2 Catalog DAGs

We briefly review the fractional cascading framework of Chazelle and Guibas. The multiple lookup query problem is modeled by a undirected graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges. The graph G is undirected and connected, and contains no loops or multiple edges. Each vertex v has a catalog C_v, and associated with each e of G is a range $e.\textsc{Range}$. A **catalog** is an ordered collection of records where each record r has a key $r.\textsc{Key} \in \mathbb{R} \cup \{\infty, -\infty\}$. The records are stored in non-decreasing order of their keys, which are not necessarily distinct. The graph G together with the associated catalogs and ranges is called a **catalog graph**. Certain nodes of G are called **gateways**. A multiple lookup query is specifed as a pair $q = (k, T)$ where $k \in \mathbb{R}$ and T is a rooted subtree of G, rooted in some gateway node. The response to this query is the set of successors of k in each catalog C_v, $v \in T$. The **successor** of any $k \in \mathbb{R}$ in a catalog C_v is the first record in C_v whose key is $\geq k$.

Critique. All actual examples of G (except for "synthetic ones" [4]) actually satisfy several other properties: first, G is a directed graph. Second, it is acyclic. Finally, it is rooted at some node that serves as the gateway. Indeed, G is most often a tree. We incorporate these aspects into our formulation to greatly simplify fractional cascading. It turns out that by making catalog graphs directed, we open the way to another important innovation of this paper (see semibridges below). Another unsatisfactory aspect of catalog graphs is the formulation of a query as a pair $q = (k, T)$. The "query tree" T is generally a complex object and is magically given. In practice, this will be dynamically constructed as part of the query. We will formalize this aspect too.

We introduce some key concepts: a **catalog digraph** $G = (V, E)$ is a rooted digraph where each $v \in V$ has a catalog C_v. The root v_0 plays the role of the gateway, and it can reach all other nodes. In case G is acyclic, we call it a **catalog DAG**. A catalog is an ordered list of records; each record has a key $r.\textsc{Key}$ and a discriminant $r.\textsc{Disc}$. A key is an element of $\mathbb{R} \cup \{\pm\infty\}$. We leave the nature of discriminants unspecified, but in practice, $r.\textsc{Disc} \in \mathbb{R}^k$ for some fixed k. The records in a catalog list are stored in non-decreasing key values. Let $\mathcal{T}$ be a finite set of symbols, called **types**. Each edge $e \in E$ has a range $e.\textsc{Range}$ and a type $e.\textsc{Type}$. The catalog is **type bounded** in the following sense: for each $v \in V$ and $x \in \mathbb{R}$, and each type $\tau \in \mathcal{T}$, there is at most one edge e emanating from v such that $x \in e.\textsc{Range}$ and $e.\textsc{Type} = \tau$. A **multiple lookup query** q is a record (so it has two fields, $q.\textsc{Key}$ and $q.\textsc{Disc}$). We further assume that there is a discriminant function $D(q, r.\textsc{Disc})$ which returns a subset of $\mathcal{T}$. Given any query q, this function uniquely determines a **query subgraph** T as follows: the root v_0 is in T. Suppose v is in T and r is the successor of $q.\textsc{Key}$ in C_v. Then the children of v in T comprise those u such that the edge $e = (v, u)$ belongs to E and $e.\textsc{Type} \in D(q, r.\textsc{Disc})$. Finally, the response to query q will be the set of successors of $q.\textsc{Key}$ in each node of T (or in each leaf of T, for another variant of the problem).

Semibridges. Chazelle and Guibas introduced the notion of "bridges" which is retained by all subsequent authors. Bridges are bi-directional pointers between two lists, and are hard to maintain. But unidirectional pointers, or "semibridges", are sufficient for most applications. Let L, L' be **blocked lists** of records, i.e., lists which are partitioned into sublists called **blocks**. The usual concept of a list is the special case where each block has size 1. A **semibridge** for $u \in L$ is a pointer $u.\textsc{Link}$ to the block in L' which contains the successor $u' \in L'$ of u. If for every $u \in L$, we maintain the pointer $u.\textsc{Link}$ to the successor block of u in L', then we write $L \Longrightarrow L'$ and say there is a **linkage** from L to L'. We write $L \Longleftrightarrow L'$ if $L \Longrightarrow L'$ and $L' \Longrightarrow L$. To support fractional cascading, we need two additional numerical invariants. Let $\text{B-}\textsc{Rank}_L(u) = \text{B-}\textsc{Rank}(u) \geq 1$ denote the position of the block containing u in L ($\text{B-}\textsc{Rank}(u) = 1$ iff u is in the first block of L). Let $g_0, g_1 > 0$ be parameters. We write $L \overset{g_0,g_1}{\Longrightarrow} L'$ if for all $u, v \in L$, if $u', v' \in L'$ are their respective successors in L', then

$$|\text{B-}\textsc{Rank}(v') - \text{B-}\textsc{Rank}(u')| \leq g_0 |1 + \text{B-}\textsc{Rank}(v) - \text{B-}\textsc{Rank}(u)|, \quad (1)$$

$$|\text{B-}\textsc{Rank}(v) - \text{B-}\textsc{Rank}(u)| \leq g_1 |1 + \text{B-}\textsc{Rank}(v') - \text{B-}\textsc{Rank}(u')|. \quad (2)$$

These are, respectively, the **forward gap invariant** and the **backward gap invariant**. For searching from L to L', we only need the forward gap invariant. If we need to modify L' then the backward gap invariant is useful. If $g_0 = g_1 = g$, then we simply write $L \overset{g}{\Longrightarrow} L'$ instead of $L \overset{g,g}{\Longrightarrow} L'$.

3 B-graph Data Structure

As in B-trees, our data structure is parameterized by two natural numbers a and b. We constraint a, b to satisfy

$$m \leq a \leq \frac{mb + 1}{m + 1}$$

for some fixed integer $m \geq 2$. This is a generalization of the usual constaint where m is assumed to be 2 (see for example [9]). Roughly speaking, the ratio a/b is the guaranteed space utilization of our data structure. With the choice $m = 2$, this ratio is $\leq 1/2$ (as $b \to \infty$), guaranteeing about 50% space utilization. As m increases, this ratio approaches 1 or 100% space utilization. E.g., $m = 4$ can ensure up to 80% utilization. The effects of these parameters on our algorithms will be described in the full paper.

Our data structure stores two types of **items**: **records** and **semibridges**. As described in connection with catalogs, a record r has two parts, $r.\textsc{Key}$ and $r.\textsc{Disc}$. It also has a data part $r.\textsc{Data}$ which is largely irrelevant for our algorithms. But in fractional cascading, we will need to duplicate records; such records have a "null" data part and are called **dummies**. A semibridge s has a pointer denoted $s.\textsc{Link}$ and a type indicator $s.\textsc{Type}$. The value $s.\textsc{Type}$ comes from some finite set $\mathcal{T}$ of "types", depending on the application. Let $|\mathcal{T}| = d$, so there are d possible types. If the type of a semibridge is τ, then the semibridge is called a τ-link. A **B-node** then is just a sequence of records, interspersed with semibridges. Between any two records, there is at most one semibridge of each type. The pointers in semibridges lead to other B-nodes. Records in a B-node are sorted by non-decreasing key value. We leave unspecified as to how this sorted order is maintained in a B-node. Each B-node has at most b records, and has, with certain exceptions, at least a records. We define a **B-List** to be a doubly-linked list of B-nodes. The B-nodes are also sorted with respect to each other: the keys in the ith B-node is no larger than those in the $i+1$st B-node.

Let L and L' be two B-Lists. Let τ be a type. For any record r in a B-node, its τ-semibridge is the first τ-semibridge that precedes r in its B-node. To ensure that r always has a τ-semibridge, the first d items in any B-node will be semibridges, one of each type. The relation $L \overset{g}{\Longrightarrow}_\tau L'$. holds between two B-lists provided:

(i) For each record r in L, its τ-semibridge points to the B-node in L' that contains its L'-successor.

(ii) The forward and back gap invariants are satisfied with the parameter g.

Observe that a B-tree (with level-links) can be regarded as a sequence of B-Lists such that

$$B_0 \overset{1}{\Longrightarrow}_0 B_1 \overset{1}{\Longrightarrow}_0 \cdots \overset{1}{\Longrightarrow}_0 B_k$$

where the type 0 is special and B_0 is constrained to have only one B-node. We define a **B-graph** to be a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ such that to each $v \in \mathcal{V}$, we associate a B-list L_v and to each $(u,v) \in \mathcal{E}$, we have a linkage, $L_u \overset{g}{\Longrightarrow}_\tau L_v$ for some $\tau \in \mathcal{T}$. Here, $\mathcal{T}$ and g are fixed by the B-graph. We say that a B-graph $\mathcal{G}$ **implements** a catalog digraph $G = (V, E)$ if there is a 1-1 map μ from V to $\mathcal{V}$ with 3 properties: (1) The catalog C_v is a subsequence of the B-list $L_{\mu(v)}$. (2) If $e = (u,v) \in E$ then $L_{\mu(u)} \overset{g}{\Longrightarrow}_\tau L_{\mu(v)}$ where $\tau = e.\textsc{Type}$. (3) $\mathcal{G}$ contains a B-tree on $L_{\mu(v_0)}$ where v_0 is the root of G.

Clearly, if $\mathcal{G}$ implements G, then for any multiple lookup query q, we can use the discriminant function D of G, together with $\mathcal{G}$, to answer the query. Assuming that the discriminant function D can be evaluated in constant time, we have the following analogue of Chazelle-Guibas's main result:

Theorem 2 *For any catalog DAG G, we can construct a B-graph $\mathcal{G}$ that implements G and which responds to any query q in time $O(td + \log_b s)$, where s is the total size of the catalogs in G, $d = |T|$ and t is the number of catalogs visited by the query q.*

4 Randomized Incremental Monotone Chain Algorithm

We describe the main complexity result, which treats the static case and is the most important situation in our map applications. The goal is to generalize the Seidel-Mulmuley algorithm for randomized incremental construction of a planar point location data structure. Instead of line segments, we randomly insert monotone chains. This requires fractional cascading to achieve optimal search time. We define catalog DAG G corresponding to this problem, and provide a construction of the B-graph $\mathcal{G}$ that implements G. This amounts to describing the algorithm for inserting a monotone chain and its subsequent cascading process. The cascading closely imitates the well-known B-tree algorithms for insertion. We give an amortized analysis for the complexity of this construction. Finally, we analyze the expected cost of point location in $\mathcal{G}$. This analysis is outlined in the introduction. Finally, we also extend these algorithms to the dynamic setting.

References

[1] P. K. Agarwal, L. Arge, G. Brodal, and J. S. Vitter. I/O-efficient dynamic point location in monotone subdivisions. In *Proc. 10th ACM-SIAM Sympos. Discrete Algorithms*, pages 11–20, 1999.

[2] R. Bayer and McCreight. Organization of large ordered indexes. *Acta Inform.*, 1:173–189, 1972.

[3] B.Chazelle and L. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1:133–162, 1986.

[4] B. Chazelle and L. J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1:163–191, 1986.

[5] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[6] H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.*, 15:317–340, 1986.

[7] S. Huddleston and K. Mehlhorn. A new data structure for representing sorted lists. *Acta Inform.*, 17:157–184, 1982.

[8] D. G. Kirkpatrick. Optimal search in planar subdivision. *SIAM J. Computing*, 12:28–35, 1983.

[9] K. Mehlhorn. *Datastructures and Algorithms 1: Sorting and Sorting*. Springer-Verlag, Berlin, 1984.

[10] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5:215–241, 1990.

[11] F. P. Preparata and R. Tamassia. Fully dynamic point location in a monotone subdivision. *SIAM J. Comput.*, 18:811–830, 1989.

[12] R. Seidel. A simple and fast incremental algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.*, 1:51–64, 1991.

[13] S. Sen. Fractional cascading revisited. *J. Algorithms*, 19(2), 1995.

[14] Topologically integrated geographic encoding and referencing system (tiger) homepage, 1995. US Census Bureau. URL http://www.census.gov/ftp/pub/geo/www/tiger/.