Kinetic heap-ordered trees: tight analysis and improved algorithms

Guilherme D. da Fonseca [a] Celina M. H. de Figueiredo [b]

[a] *COPPE, Universidade Federal do Rio de Janeiro, Caixa Postal 68530, 21945-970 Rio de Janeiro, RJ, Brazil.* gfonseca@esc.microlink.com.br

[b] *Instituto de Matemática and COPPE, Universidade Federal do Rio de Janeiro, Caixa Postal 68530, 21945-970 Rio de Janeiro, RJ, Brazil.* celina@cos.ufrj.br

Abstract

The most natural kinetic data structure for maintaining the maximum of a collection of continuously changing numbers is the kinetic heap. Basch, Guibas, and Ramkumar proved that the maximum number of events processed by a kinetic heap with n numbers changing as linear functions of time is $O(n \log^2 n)$ and $\Omega(n \log n)$. We prove that this number is actually $\Theta(n \log n)$. In the kinetic heap, a linear number of events are stored in a priority queue, consequently, it takes $O(\log n)$ time to determine the next event at each iteration. We also present a modified version of the kinetic heap that processes $O(n \log n / \log \log n)$ events, with the same $O(\log n)$ time complexity to determine the next event.

Key words: kinetic data structures, heaps, computational geometry, data structures

1 Introduction

Basch, Guibas, and Hershberger [1] introduced the concept of kinetic data structures by presenting two structures for maintaining the maximum of a collection of continuously changing numbers: the kinetic (binary) heap and the kinetic tournament. They argued that counting the number of events of the kinetic heap was surprisingly non-trivial and proved that, if the n stored numbers change as linear functions of time, then the kinetic tournament only processes $O(n \log n)$ events. Basch, Guibas, and Ramkumar [2] proved that, if the n stored numbers change as linear functions of time, then the kinetic heap only processes $O(n \log^2 n)$ and $\Omega(n \log n)$ events in the worst case. In [2,3] it was asked whether the kinetic heap actually processes $O(n \log n)$ events in the worst case. We prove that this is the case, by proving a stronger result: In any *kinetic heap-ordered tree*, the maximum number of events processed is *exactly* the number of edges in the transitive closure of the tree (we consider the edges directed from the root to the leaves). As this number of edges is $O(n \log n)$ for a tree of height $O(\log n)$, the claimed upper bound follows.

A *heap-ordered tree* is a rooted tree where the value of the element associated with each node is greater than the value of the elements associated with its children. In a *kinetic heap-ordered tree*, each element is a continuous real function of time. As usual [1,2], we restrict ourselves to linear functions. As the values of the functions change, the corresponding elements have to move to different nodes in the tree so that the rooted tree is kept heap-ordered. A move occurs when, for any edge (v_1, v_2) with associated elements f_1 and f_2, value $f_1(t)$ becomes equal to value $f_2(t)$. To keep the rooted tree heap-ordered, we exchange the elements associated with nodes v_1 and v_2 at that moment. We call the instants of time when we have to update the tree *events*. To keep track of the events, we maintain a priority queue containing the edges that may need to be changed. This priority queue is the *event queue* of the kinetic structure.

To make our terminology shorter, we will not distinguish between a node, the element associated with it and its value, unless absolutely necessary. For example, we can say "each node is greater than its children" instead of "the value of the element associated with each node is greater than the value of the element associated with its children", or "the descendants of the element f with slopes greater than the slope of f" instead of "the elements whose slopes are greater than the slope of the element f, and which are associated with nodes that are descendants of the node associated with f".

A *kinetic binary heap* is a kinetic heap-ordered tree where the tree is a full binary tree. A *kinetic k-heap* is a kinetic heap-ordered tree where the tree a is full k-ary tree.

In a kinetic binary heap with n elements, each event can be processed in $O(\log n)$ time. This is true because we need to keep at most $n - 1$ events in the event queue at a time (one for each edge) and a constant number of events will have to be rescheduled at each event. But how many events will a kinetic heap process in the worst case? This question remained open since the introduction of the kinetic heaps [1] and is answered in the next section.

2 Number of Events

Given a kinetic heap-ordered tree K, let $\Phi(K)$ be the number of edges in the transitive closure K^* of K (consider the edges directed from the root to the leaves). There are two natural ways to calculate $\Phi(K)$: the sum of the out-degree in K^* of all nodes, and the sum of the in-degrees in K^* of all nodes. The former is the sum of the number of descendants in K of all nodes, and the latter is the sum of the levels in K of all nodes (root with level 0). Throughout the paper, we alternate between these two approaches, choosing the most convenient one. The former is referenced as top-down and the latter as bottom-up.

We prove that, for any kinetic heap-ordered tree K, the number of events is exactly $\Phi(K)$ in the worst case. The height of a kinetic binary heap K with n elements is $O(\log n)$ and $\Phi(K) = O(n \log n)$, which imply the claimed upper bound for the number of events. First we prove $\Phi(K)$ is a lower bound for the number of events. The proof is essentially the same as in [2].

Lemma 1 *If the n linear functions of time stored in a kinetic heap-ordered tree K are all tangent to the parabola $y = t^2$, at least $\Phi(K)$ events will be processed if we start with a sufficiently small value of t.*

PROOF. Each element will be the root of the tree at some time t. At each event, only one element goes up towards the root by moving exactly one level. Thus the level of a node gives the number of events needed to take this node to the root. Using the bottom-up definition of $\Phi(K)$, it easy to see that $\Phi(K)$ events are necessary to take every element of the tree to the root. $\square$

To prove that $\Phi(K)$ is also an upper bound we define below another function $\Phi'(K)$, satisfying $\Phi'(K) \leq \Phi(K)$. This function $\Phi'(K)$ depends not only on the structure of the tree, but also on the elements of K and the nodes they are associated with. Because of that, we refer to the kinetic heap-ordered tree K at time t as K_t and deal with $\Phi'(K_t)$ instead of $\Phi'(K)$. If an event occurs at time t, we refer to the kinetic heap-ordered tree K just before and just after the nodes are exchanged at time t as K_{t-} and K_{t+}, respectively. We shall prove that $\Phi'(K_t)$ is an upper bound to the number of events processed by a kinetic heap-ordered tree K after time t.

For every element f of K_t, let $\Phi'_{\downarrow}(K_t, f)$ be the number of descendants of f that will become greater than f for some $t' \geq t$ (the ones with a slope greater than the slope of f). We define

$$\Phi'(K_t) = \sum_{f \in K_t} \Phi'_{\downarrow}(K_t, f).$$

For every element f of K_t, for every child c of f in K_t, let $\Phi'_{\downarrow}(K_t, f, c)$ denote the number of elements in the sub-tree rooted at c that will become greater than f for some $t' \geq t$. We can compute $\Phi'_{\downarrow}(K_t, f)$ as the following sum:

$$\Phi'_{\downarrow}(K_t, f) = \sum_{c \text{ child of } f} \Phi'_{\downarrow}(K_t, f, c).$$

Lemma 2 *The number of events processed in a kinetic heap-ordered tree K after time t is at most $\Phi'(K_t)$.*

PROOF. We shall prove that, if an event is processed at time t, $\Phi'(K_{t+}) \leq \Phi'(K_{t-}) - 1$. Since $\Phi'(K_t) \geq 0$, for all t, that will prove the lemma.

Suppose an event occurred at time t because $p(t) = c_1(t)$ and p is the parent of c_1 in K_{t-}. We have that

$$\Phi'_{\downarrow}(K_{t+}, f) = \Phi'_{\downarrow}(K_{t-}, f), \text{ for all } f \neq p \text{ and } f \neq c_1.$$

Moreover,

$$\Phi'_{\downarrow}(K_{t+}, p) = \Phi'_{\downarrow}(K_{t-}, p, c_1) - 1.$$

Let $c_1, c_2, \ldots, c_q$ be the children of p in K_{t-}. Since the slope of c_1 is greater than the slope of p, we have

$$\Phi'_{\downarrow}(K_{t+}, c_1) \leq \Phi'_{\downarrow}(K_{t-}, c_1) + \sum_{i=2}^{q} \Phi'_{\downarrow}(K_{t-}, p, c_i).$$

Adding these equations we have $\Phi'(K_{t+}) \leq \Phi'(K_{t-}) - 1$. $\square$

As $\Phi'(K) \leq \Phi(K)$, our result now follows immediately from Lemma 1 and Lemma 2:

Theorem 3 *The maximum number of events processed in a kinetic heap-ordered tree K is exactly $\Phi(K)$.*

Corollary 4 *The maximum number of events processed in a kinetic binary heap with n elements is $\Theta(n \log n)$.*

Lemma 2 bounds the total number of events processed in a kinetic heap-ordered tree, but it does not answer another natural question: How many times, in the worst case, does the node associated with an element f change? To answer this question we have to calculate $\Phi'(K)$ bottom-up.

For every element f of K_t, let $\Phi'_{\uparrow}(K_t, f)$ be the number of ancestors of f that will become less than f for some $t' \geq t$ (the ones with a slope less than the slope of f). It is easy to see that $\Phi'(K_t)$ can also be computed as the sum:

$$\Phi'(K_t) = \sum_{f \in K_t} \Phi'_{\uparrow}(K_t, f).$$

Theorem 5 *If f is an element of K_t, then f goes one level up towards the root at most $\Phi'_{\uparrow}(K_t, f)$ times after time t.*

PROOF. We shall prove that, if an event is processed at time t, then, for all elements f, $\Phi'_{\uparrow}(K_{t+}, f) \leq \Phi'_{\uparrow}(K_{t-}, f)$, and that, if f is the element which goes up one level towards the root at time t, then $\Phi'_{\uparrow}(K_{t+}, f) = \Phi'_{\uparrow}(K_{t-}, f) - 1$. Since $\Phi'_{\uparrow}(K_t, f) \geq 0$, for all t and f, that will prove the lemma.

Suppose an event occurred at time t because $p(t) = c(t)$ and p is the parent of c in K_{t-}. Let S be the set of elements contained in K. The elements of $S - \{p, c\}$ can be partitioned into the sets:

$S_1 = \{f \in S - \{p, c\} : f$ is *not* a descendant of p in $K_{t-}\}$

$S_2 = \{f \in S - \{p, c\} : f$ is a descendant of c in $K_{t-}\}$

$S_3 = \{f \in S - \{p, c\} : f$ is a descendant of p but *not* of c in $K_{t-}\}$

If $f \in S_1 \cup S_2$, then the ancestors of f in K_{t-} and K_{t+} are the same. Consequently, for all $f \in S_1 \cup S_2$, we have $\Phi'_{\uparrow}(K_{t+}, f) = \Phi'_{\uparrow}(K_{t-}, f)$. If $f \in S_3$, then the ancestors of f changed from K_{t-} to K_{t+} by the removal of p and addition of c. As the slope of c is greater than the slope of p, we have $\Phi'_{\uparrow}(K_{t+}, f) \leq \Phi'_{\uparrow}(K_{t-}, f)$, for all $f \in S_3$. It is immediate that $\Phi'_{\uparrow}(K_{t+}, p) = \Phi'_{\uparrow}(K_{t-}, p)$ and $\Phi'_{\uparrow}(K_{t+}, c) = \Phi'_{\uparrow}(K_{t-}, c) - 1$. $\square$

3 Improvements

In Section 1, we said that the number of events stored at the same time in the event queue of a kinetic heap with n elements is at most $n - 1$, which is the number of edges in the tree. We can speed up a kinetic heap with a simple modification. If $c_1, c_2, \ldots, c_m$ are children of p in a kinetic heap, we only schedule an event (p, c_j) for the child c_j that will first become greater than p. It is clear that scheduling events for the other children of p is a waste of time, because those events will be rescheduled when we process the event for the edge (p, c_j).

This modification does not change the asymptotic time complexity of a kinetic binary heap, but allows us to build an efficient kinetic $\log n$-heap. We analyze a generic kinetic $f(n)$-heap. In a generic kinetic $f(n)$-heap, let $V_{f(n)}$ denote the maximum number of events processed and $T_{f(n)}$ denote the time required to process each event. By Theorem 3, a kinetic $f(n)$-heap processes $V_{f(n)} = \Theta(n \log_{f(n)} n) = \Theta(n \log n / \log f(n))$ events, in the worst case. The time to process an event, using the strategy described in the previous paragraph, is $T_{f(n)} = \Theta(f(n) + \log(n/f(n))) = \Theta(f(n) + \log n)$, where $\Theta(f(n))$ is the time spent examining all the children of the affected nodes and $\log(n/f(n))$ is the time spent accessing a priority queue with $n/f(n)$ events.

In a kinetic binary heap, $f(n) = 2$ is constant. As we use faster growing functions for $f(n)$, the number of events processed decreases together with the height of the tree, but the time to process each event increases. Normally, we desire to minimize $V_{f(n)}T_{f(n)}$, which is the time complexity of processing all the events in the worst case. This minimum is attained with $f(n) = \Theta(\log n)$, for which $V_{\log n}T_{\log n} = \Theta(n \log^2 n / \log \log n)$.

4 Conclusion and Open Problems

If the elements are linear functions of time, the number of events processed in the worst case by a kinetic binary heap and other heap-ordered trees is completely determined. This analysis improves the one given in [2] by a factor of $\log n$ and is tight. Actually, our analysis is still valid if the set of elements is a set of *pseudo-lines*. A set of continuous real functions $\{f_1, f_2, \ldots, f_n\}$ is a set of *pseudo-lines* if, for each pair $i \neq j$, $f_i(t) = f_j(t)$ for at most one value of $t = t_{ij}$, and either $f_i(t) < f_j(t)$ when $t < t_{ij}$ and $f_i(t) > f_j(t)$ when $t > t_{ij}$ or $f_j(t) < f_i(t)$ when $t < t_{ij}$ and $f_j(t) > f_i(t)$ when $t > t_{ij}$.

We did not mention some natural operations in a kinetic heap: the insertion of an element, the deletion of an element, and the change of the flight plan of an element. When changing the flight plan, we replace, at time t, an element f_1 by another element f_2 such that $f_1(t) = f_2(t)$. The number of events processed by a kinetic heap under these operations is largely unknown. If we start with an empty heap and make n insertions and n deletions, the number of events is $O(n\sqrt{n \log n})$ [2]. We believe this bound is not tight, and the only lower bound known is $\Omega(n \log n)$.

Another intriguing question is the number of events processed by a kinetic heap where the elements are non-linear functions of time. No result is known.

Although the improvement we suggest in Section 3 results in a reasonable speed-up in practice, the asymptotic improvement is small. A kinetic $\log n$-

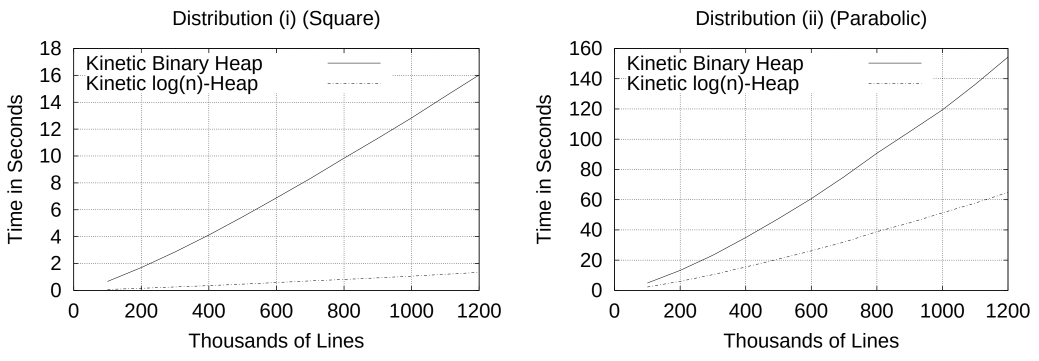

Fig. 1. Times to compute the upper envelope of sets of lines on a 1GHz Athlon

heap can be used to compute the upper envelope of a set of lines (or its geometric dual, the convex hull) in time $\Theta(n \log^2 n / \log\log n)$. This is an improvement to the algorithm to compute the upper envelope described in [2] and the *heaphull* algorithm described in [3], but many well known algorithms compute the upper envelope/convex hull in optimal $O(n \log n)$ time.

We have implemented a kinetic k-heap and tested it computing the upper envelope for $t \geq 0$ of a set lines $y = at + b$ using the following probability distributions: (i) a and b are independent random variables in the interval $[0, 1]$; (ii) r is a random variable in the interval $[0, 1]$, and $a = 2r$ and $b = -r^2$. Case (ii) is the worst case for the number of events described in Section 2, where the lines are tangent to the parabola $y = t^2$. For 10^6 lines, our implementation of the kinetic $\log n$-heap is faster than the kinetic binary heap by a factor of approximately 12 in case (i), and 2.3 in case (ii) (see Figure 1). We do not know any way to probabilistically calculate this discrepancy between the two average case times.

Acknowledgements

We thank referees for their careful reading and valuable suggestions, which helped improve the presentation of this paper.

References

[1] J. Basch, L. J. Guibas, J. Hershberg, Data Structures for Mobile Data, in Proc. 8th Annual ACM-SIAM Symposium on Discrete Algorithms, 1997, pages 747-756; Journal of Algorithms, Vol. 31, No. 1, April 1999, pages 1-28.

[2] J. Basch, L. J. Guibas, and G.D. Ramkumar, Sweeping lines and line segments with a heap, in Proc. 13th Annual ACM Symposium on Computatinal

Geometry, 1997, pages 469-471.

[3] Andrea Mantler, Jack Snoeyink, Heaphull?, in Proc. 13th Canadian Conference on Computational Geometry, 2001, pages 129-131.