Kinetic hanger

Guilherme D. da Fonseca [a] Celina M. H. de Figueiredo [b]
Paulo C. P. Carvalho [c]

[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

[c] *Instituto de Matemática Pura e Aplicada, Estrada Dona Castorina 110, 22460-320 Rio de Janeiro, RJ, Brazil.* pcezar@impa.br

Abstract

A kinetic priority queue is a kinetic data structure which determines the largest element in a collection of continuously changing numbers subject to insertions and deletions. Due to its importance, many different constructions have been suggested in the literature, each with its pros and cons. We propose a simple construction that takes advantage of randomization to achieve optimal locality and the same time complexity as most other efficient structures.

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

1 Kinetic Priority Queues

A *kinetic priority queue* is a *kinetic data structure* which computes the largest element in a collection of continuously changing numbers subject to insertions and deletions. Kinetic priority queues are the most fundamental kinetic data structures and have been applied not only to solve kinetic problems, but also to solve the famous k-set problem [6] and the connected red blue segments intersection problem [2]. Due to its importance, many different constructions have been suggested: the kinetic heap [4], the kinetic heater [2], the kinetic tournament [4], and a reduction to dynamic convex hull structures [6]. Each of these structures has its pros and cons.

The most important theoretical evaluation of kinetic priority queues consists of comparing their time complexities on a real random access machine (real-

RAM). We define the time complexity of a kinetic priority queue in a given scenario as the total processing time of the structure sweeping the scenario. This sweep line algorithm, computes the maximum of a set of functions of time along time, that is, the upper envelope of these functions. When the functions are only defined within an interval of time, we call these functions segments. Each time a new segment appears, we perform an insertion and each time a segment ceases to exist, we perform a deletion. It is important to notice that this is different from solving the static upper envelope problem because we do not assume the scenario is completely known to the right of the sweep line. We assume that our model of computation can calculate the exact intersection of two real functions in $O(1)$ time. The time complexities of the structures in the literature are summarized in Table 1. We use lg to denote the binary logarithm, $\alpha(\cdot)$ to denote the inverse Ackerman function and $\lambda_\delta(n)$ to denote the maximum size of the upper envelope of n δ-intersecting curves. We use n/a to denote not available. For details on $\lambda_\delta(n)$, see [11].

#	Scenario	Heap	Heater, Hanger and Tournament	Dynamic Hull
1	Lines	$O(n \lg^2 n)$	$O(n \lg^2 n)$	$O(n \lg n)$
2	Line segments	$O(m\sqrt{n} \lg^{3/2} n)$	$O(m\alpha(n) \lg^2 n)$	$O(m \lg n \lg \lg n)$
3	δ-intersecting curves	$O(n^2 \lg n)$	$O(\lambda_\delta(n) \lg n)$	n/a
4	δ-intersecting curve segments	$O(mn \lg n)$	$O(m/n\lambda_{\delta+2}(n) \lg n)$	n/a

Table 1
Time complexities of the structures. Parameter δ is considered constant. In scenarios 2 and 4, parameter n denotes the maximum number of elements stored in the structure at the same time and m denotes the total number of elements. The dynamic hull constructions do not support curves.

A *heap* is a binary tree, where the priority of each node is greater than the priority of its children. We call this property the *heap property*. The heap is balanced in the sense that all nodes with less than two children are in the last two levels of the tree. It is not hard to perform insertions and deletions in a heap, and details can be found in [8]. In a *kinetic heap* the priorities change continuously with time. We call the moment when the priority of an internal node becomes equal to the priority of one of its children an *event*. At an event, the structure of the heap must be changed, in order to maintain the heap property. To find events efficiently, we store potential events in a efficient priority queue (like a heap), which is called the *event queue*. For all internal nodes, we store a corresponding potential event, scheduled to the time when the priority of the node would become equal to the priority of one of its children if the structure of the heap remained unchanged. To advance in time, we process the first occurring potential event in the event queue, which

is clearly an event. Processing an event consists of swapping the contents of the two nodes involved in the event, and rescheduling at most three adjacent potential events. We say that a kinetic heap has $O(1)$ locality because a constant number of potential events must be rescheduled when processing an event. Consequently, processing an event in a kinetic heap with n elements can be done in $O(\lg n)$ time. But determining the maximum number of processed events is a surprisingly nontrivial problem, which has only been solved tightly when the priorities are linear functions of time (actually they only need to be 1-intersecting curves) and no insertion or deletion is performed [9]. When insertions and deletions are performed, a non-trivial bound is known [3], but it is not known whether it is tight. The time complexity of the heap with other curves is completely unknown, except for trivial results.

The kinetic *heater* [2] is a variation of the kinetic heap that uses randomization in a way similar to the treap [1], which makes it easier to analyze. A kinetic heater is also a binary tree with the heap property, but not necessarily full. In a kinetic heater, each node has not only a priority, but also a random key, and the tree is a binary search tree on the keys of the nodes. Compared to the kinetic heap, the kinetic heater is less efficient in some aspects. It requires storing the random keys and, instead of simply swapping nodes, we have to perform rotations to insert or delete an element or process an event. Besides that, variations of the kinetic heap and the kinetic tournament have been proposed in [9] and [6], respectively, where the tree is a $(\lg n)$-ary tree, instead of a binary tree. These variations reduce the time complexities of the structures by a factor of $\Theta(\lg \lg n)$. The binary search tree structure of the kinetic heater makes it hard to use the same strategy to improve its efficiency. Nevertheless, the kinetic heater has $O(1)$ locality and the number of events is tightly bounded [2].

The kinetic *tournament* [4] is another kinetic priority queue. The kinetic tournament is significantly different from the other structures, because the elements are stored only in the leaves of the tree. Each internal node is associated with the maximum element among its descendants. When processing an event, we possibly have to reschedule many events, in a path toward the root. Thus, a kinetic tournament has $O(\lg n)$ locality, even though its time complexity is the same as the heater.

Reducing the problem to a dynamic hull structure [6] is not a practical alternative, because those structures are very complex. Besides that, this reduction is restricted to the case where the priorities are linear functions of time. Nevertheless, this approach leads to the most efficient structures, having an important theoretical value. The most efficient dynamic hull structure known is presented in [5].

We would like to have a kinetic priority queue which were as simple and

```
hang(element e, node v)
{
      if v.elem = ε
            v.elem ← e
            return
      r ← randombit()
      hang (e, v.child_r)
}
```

Fig. 1. The hang procedure used to construct an initial hanger.

efficient in practice as the kinetic heap, were at least as efficiente as the kinetic heater and the kinetic tournament, and had $O(1)$ locality. We propose a new randomized kinetic priority queue, the kinetic hanger, which is both simpler and more efficient in practice than the kinetic heater, and has $O(1)$ locality.

2 Kinetic Hanger

A hanger is a randomized priority queue which is very similar to the heap. Like the heap, a hanger is a binary tree such that the priority of each node is greater than the priority of its children, but the hanger uses randomization to balance the tree. To construct a hanger with an initial set of elements we sort the elements by decreasing priorities and *hang* them at the root. We now describe the *hang* procedure. When hanging an element e at a node v we first check if there is already an element associated with node v. If there is no element associated with node v we associate e with v and return. Otherwise, we use a random bit r to choose a child c_r of v and recursively hang e at node c_r. This procedure is described in pseudo-code in Figure 1.

The kinetic hanger with a set of elements S, denoted by $hanger(S)$, is a random structure. Any binary tree h with the set of vertices S which satisfies the heap property is a possible $hanger(S)$, that is, $\Pr(hanger(S) = h) > 0$. According to the values of the random bits, different kinetic hangers may be constructed, with different probabilities. The following lemma shows how to calculate the probability of a random hanger being equal to a given tree h, which is a possible $hanger(S)$. We define L_v as the level of the vertex v in the tree h, that is, the number of edges in the path from v to the root of h.

Lemma 1 *If h is a possible $hanger(S)$, then*

$$\Pr(hanger(S) = h) = \left(\frac{1}{2}\right)^{\sum_{v \in h} L_v}.$$

```
insert(element e, node v)
{
      if v.elem = ε
            v.elem ← e
            return
      if v.elem < e
            exchange e and v.elem
      r ← randombit()
      insert (e, v.child_r)
}
```

Fig. 2. The insert procedure.

PROOF. Let R denote the sequence of bits returned by the *randombit* function in the *hang* procedure building h. It is clear that $hanger(S) = h$ if and only if the sequence of bits returned by the *randombit* function building $hanger(S)$ is exactly R, which happens with probability $2^{-|R|}$. As $|R|$, the number of bits in R, is equal to the sum of the levels of all vertices of h, the lemma is true. □

We want to define insert and delete operations that preserve the randomness of the structure in the sense that the kinetic hanger $hanger(S)$ constructed from scratch with a set of elements S and the kinetic hanger obtained by any sequence of insertions and deletions which results in the set S are equally distributed.

The insert procedure is very similar to the hang procedure. The only difference is that when we insert an element e at node v which is already associated with an element e' we compare the priorities of e and e'. If the priority of e is higher than the priority of e', instead of recursively hanging e at a random child of v we associate e with node v and recursively hang e' at a random child of v. This procedure is described in Figure 2.

We denote by $insert(h, e)$ the hanger obtained by inserting an element e on the hanger h with the insert procedure.

Lemma 2 *Insertion preserves randomness, that is* $\Pr(insert(hanger(S), e) = h) = \Pr(hanger(S \cup e) = h)$.

PROOF. Notice that, for a given hanger h, there is only one possible hanger h' such that $insert(h', e) = h$. The only difference between the topology of h and h' is the existence of a new leaf l in h, which was not present in h'.

Consequently,

$$\begin{aligned}
&\Pr(insert(hanger(S), e) = h) \\
&= \Pr(insert(hanger(S), e) = h | hanger(S) = h')\ \Pr(hanger(S) = h') \\
&+ \Pr(insert(hanger(S), e) = h | hanger(S) \neq h')\ \Pr(hanger(S) \neq h') \\
&= \Pr(insert(hanger(S), e) = h | hanger(S) = h')\ \Pr(hanger(S) = h')
\end{aligned}$$

The probability that the insertion of e in h' causes the specific leaf l to appear is 2^{-L_l}. Using this and lemma 1,

$$\begin{aligned}
&\Pr(insert(hanger(S), e) = h | hanger(S) = h')\ \Pr(hanger(S) = h') \\
&= \left(\frac{1}{2}\right)^{L_l} \left(\frac{1}{2}\right)^{\sum_{v \in h'} L_v} = \left(\frac{1}{2}\right)^{\sum_{v \in h} L_v} \\
&= \Pr(hanger(S \cup e) = h).
\end{aligned}$$

□

Deleting an element in the hanger is also simple. Actually, it is simpler than deleting an element in the heap itself. We start from the element that we want to remove and go down replacing the current element by its child with highest priority. No extra random bits are necessary. We denote by $delete(h, e)$ the hanger obtained by deleting an element e from the hanger h with the delete procedure.

Lemma 3 *Deletion preserves randomness, that is* $\Pr(hanger(S \setminus e) = h) = \Pr(delete(hanger(S), e) = h)$.

PROOF. By Lemma 2, we can replace $hanger(S)$ by $insert(hanger(S \setminus e), e)$. Consequently,

$$\Pr(delete(hanger(S), e) = h) = \Pr(delete(insert(hanger(S \setminus e), e), e) = h)$$

Let $h' = hanger(S \setminus e)$. By definiton, $delete(insert(h', e), e) = h'$. Consequently, we can replace $delete(insert(h', e), e)$ by h', to get

$$\Pr(delete(insert(hanger(S \setminus e), e), e) = h) = \Pr(hanger(S \setminus e) = h).$$

□

Clearly, in a hanger with n elements, the hang, insert and delete procedures have expected time complexity of $O(E(L_n))$.

Lemma 4 *$E(L_n) \leq \lg n$. Besides that, $\Pr(L_n \geq \lg n + c) \leq 2^{-c}$.*

PROOF. Instead of bounding $E(L_n)$ directly, we prove $E(2^{L_n}) = n$. By Jensen's inequality, $2^{E(x)} \leq E(2^x)$ and the lemma follows.

Let r denote the root of a hanger with n elements. Let r_l, r_r denote the two subtrees rooted at the left and right children of r, respectively, and $|r_l|$ and $|r_r|$ denote the number of vertices in these subtrees. Let r_n denote the tree, either r_l or r_r, which contains the element n. If $|r_n| = i$, then $E(2^{L_n})$ is two times $E(2^{L_{i+1}})$. Consequently,

$$E\left(2^{L_n}\right) = \sum_{i=0}^{n-2} \Pr(|r_n| = i) 2E\left(2^{L_{i+1}}\right).$$

The number of elements on r_l is a binomial random variable ranging from 0 to $n-1$. The same holds for r_r. The number of elements on r_n is 1 plus a binomial random variable ranging from 0 to $n-2$. Using this, we can construct the following probabilistic recurrence:

$$E\left(2^{L_n}\right) = \sum_{i=0}^{n-2} \binom{n-2}{i} \frac{1}{2^{n-2}} 2E\left(2^{L_{i+1}}\right).$$

To prove that $E(2^{L_n}) = n$ we use induction. By applying the induction hypothesis and the absorption property of binomial coefficients [10], namely

$$k\binom{n}{k} = \binom{n-1}{k-1} n, \text{ we obtain:}$$

$$\begin{aligned}
E(2^{L_n}) &= \sum_{i=0}^{n-2} \binom{n-2}{i} \frac{1}{2^{n-3}} (i+1) \\
&= \frac{1}{2^{n-3}} \left(\sum_{i=0}^{n-2} \binom{n-2}{i} + \sum_{i=1}^{n-2} \binom{n-2}{i} i \right) \\
&= \frac{1}{2^{n-3}} \left(2^{n-2} + (n-2) \sum_{i=0}^{n-3} \binom{n-3}{i} \right) \\
&= \frac{1}{2^{n-3}} \left(2^{n-2} + (n-2) 2^{n-3} \right) \\
&= n.
\end{aligned}$$

Now, we use Markov's inequality to bound the probability that L_n is much

greater than $\lg n$:

$$\Pr(L_n \geq \lg n + c) = \Pr(2_n^L \geq n2^c) \leq 2^{-c}.$$

□

This finishes the analysis of a static hanger. Now we can define and analyze the kinetic hanger. We maintain an event queue and process events in the same way as in the kinetic heap. When the priority of a node becomes equal to the priority of one of its children, we process an event, exchanging the two nodes involved in the event and rescheduling at most three adjacent events.

We denote by $hanger_t(S)$ the kinetic hanger with a set of elements S, constructed at time t. We are interested in the case where two elements intersect at time t and denote by $t-$ the moment of time just before the intersection and by $t+$ the moment of time just after the intersection.

It is also not hard to prove that intersection preserves randomness:

Lemma 5 *Intersection preserves randomness. More precisely, let* $h = hanger_{t-}(S)$. *If the two elements which intersect at time* t *are* not *parent/child in* h *then* $\Pr(hanger_{t-}(S) = h) = \Pr(hanger_{t+}(S) = h)$. *If the two elements which intersect at time* t *are parent/child in* h *and* h' *is the hanger with these two elements interchanged, then* $\Pr(hanger_{t-}(S) = h) = \Pr(hanger_{t+}(S) = h')$.

PROOF. The probability that $hanger(S) = h$ is calculated in Lemma 1, and does not depend on the values of the elements. Consequently, the lemma follows. □

Up to now, all results we obtained for the kinetic hanger have equivalent results which are valid for the kinetic heap in a deterministic worst-case fashion. Now, we are going to address the problem of calculating the expected number of events in a kinetic hanger, which has no equivalent worst-case result in the heap. Before that, we prove a simple probabilistic lemma.

Lemma 6 *Let* a, b, c *be independent, identically distributed discrete random variables. Then* $\Pr(a = b | a \neq c) \leq \Pr(a = b)$.

PROOF. Observe that $\Pr(a = b | a \neq c) = \Pr(a = b \text{ and } a \neq c)/\Pr(a \neq c) = \Pr(a = b \neq c)/1 - \Pr(a = b)$.

Hence, claim is equivalent to $\Pr(a = b \neq c) \leq \Pr(a = b) - \Pr(a = b)^2 = \Pr(a = b \neq c) + \Pr(a = b = c) - \Pr(a = b)^2$ and therefore also to $\Pr(a = b)^2 \leq \Pr(a = b = c)$.

Now observe that $\Pr(a = b)^2 = (\sum_i p_i^2)^2$ and $\Pr(a = b = c) = \sum_i p_i^3$.

Now recall Cauchy-Schwarz inequality: $(\sum_i x_i y_i)^2 \leq (\sum_i x_i^2)(\sum_i y_i^2)$. Setting $x_i = p_i^{3/2}$ and $y_i = p_i^{1/2}$, we deduce: $(\sum_i p_i^2)^2 \leq (\sum_i p_i^3)(\sum_i p_i) = (\sum_i p_i^3)$, as desired. □

Now we can prove that:

Lemma 7 *Let h be a random hanger with $m > n$ elements $f_1 > f_2 > \ldots > f_m$. The probability p_n that f_{n+1} is a child of f_n in h is $O(1/n)$.*

PROOF. We denote the event that f_n is a child of f_{n-1} in h by ϵ_n. Conditioning yields:

$$Pr(\epsilon_n) = Pr(\epsilon_n|\epsilon_{n-1})Pr(\epsilon_{n-1}) + Pr(\epsilon_n|\overline{\epsilon_{n-1}})(1 - Pr(\epsilon_{n-1})).$$

We denote by $b(f_n)$ the sequence of random bits that defines the position of f_n in h; in other words, the sequence of left/right children which defines the path from the root to f_n. We denote by $|b(f_n)|$ the size of $b(f_n)$, that is, the distance from the root to $f(n)$. Clearly, given that $|b(f_n)| = L$, $Pr(\epsilon_n) = 2^{-L}$. As a consequence, we have $Pr(\epsilon_n|\epsilon_{n-1}) = Pr(\epsilon_{n-1})/2$ because, if $n-1$ is a child of $n-2$, then $|b(f_{n-1})| = |b(f_{n-2})| + 1$.

We also claim that $Pr(\epsilon_n|\overline{\epsilon_{n-1}}) \leq Pr(\epsilon_{n-1})$. To justify that we start with a fixed random hanger h' containing elements f_1 to f_{n-3} and insert f_{n-2}, f_{n-1} and f_n. We will define three *independent equally distributed* random variables x_1, x_2 and x_3, in such a way that the event ϵ_{n-1} is the event $x_1 = x_2$ and the event ϵ_n is contained in the event $x_2 = x_3$. First, we define $x_1 = b(f_{n-2})$. To define x_2 and x_3, we truncate $b(f_{n-1})$ and $b(f_n)$, possibly removing the last one or two bits, at the point where the paths leading to f_{n-1} and f_n, respectively, leave h'. This means that we are using, for those truncated versions, the same stopping rule used to define $x_1 = b(f_{n-2})$. Thus, if we denote the truncated values of $b(f_{n-1})$ and $b(f_n)$ by x_2 and x_3, respectively, we have that x_1, x_2 and x_3 are independent and equally distributed. Now, ϵ_{n-1} holds if, and only if, the truncated version x_2 of $b(f_{n-1})$ is exactly the same as $x_1 = b(f_{n-2})$; on the other hand, if ϵ_n holds, then the truncated versions x_2 and x_3 of $b(f_{n-1})$ and $b(f_n)$ must agree.

Therefore, we have:

$$Pr(\epsilon_n|\overline{\epsilon_{n-1}}) \leq Pr(x_2 = x_3|x_1 \neq x_2) \leq Pr(x_1 = x_2) = Pr(\epsilon_{n-1}).$$

Using the above inequality, we find the recurrence:

$$Pr(\epsilon_n) \leq Pr(\epsilon_{n-1})^2/2 + Pr(\epsilon_{n-1})(1 - Pr(\epsilon_{n-1})) = Pr(\epsilon_{n-1}) - Pr(\epsilon_{n-1})^2/2.$$

Now we can prove by induction that $Pr(\epsilon_n) \leq 2/n$. It is trivially true for the base case $n = 2$. Suppose it is true for $n - 1$, then:

$$Pr(\epsilon_n) \leq Pr(\epsilon_{n-1}) - Pr(\epsilon_{n-1})^2/2 \leq 2/(n-1) - 2/(n-1)^2 =$$

$$(2n-4)/(n-1)^2 \leq 2(n-2)/n(n-2) = 2/n.$$

□

Using exactly the same argument used in [2], which is based on a theorem of Clarkson and Shor [7] we conclude:

Theorem 8 *The expected number of events processed by a kinetic hanger sweeping an arrangement Λ of n δ-intersecting curve segments is $O(\lambda_{\delta+2}(n) \lg n)$. An extra $O(\lg n)$ time must be spent processing each event.*

PROOF. We define the level l in an arrangement of functions of time as the sequence of curve segments which are the l largest segments in the arrangement. Let us denote by k_l the number of vertices at level l in the arrangement Λ. By Lemma 7, and using linearity of expectation, the expected number of events is at most

$$\sum_{l=1}^{n-1} k_l L_l = \sum_{l=1}^{n-1} k_l O(1/l).$$

Using summation by parts, we replace k_l by its partial sum $K_l = \sum_{1 \leq i \leq l} k_i$. In [7], Clarkson and Shor proved that $K_l = O(l\lambda_{\delta+2}(n))$. A standard calculation gives:

$$\begin{aligned}\sum_{l=1}^{n-1} k_l O(1/l) &= K_l O(1/l)\Big|_1^n + \sum_{l=1}^{n-1} K_{l+1} O(1/l^2) \\ &= O(\lambda_{\delta+2}(n)) + \sum_{l=1}^{n-1} O(\lambda_{\delta+2}(n)/l) \\ &= O(\lambda_{\delta+2}(n)) + O(\lambda_{\delta+2}(n)) \sum_{l=1}^{n-1} O(1/l) \\ &= O(\lambda_{\delta+2}(n) \lg n).\end{aligned}$$

□

Some useful corolaries are:

Corollary 9 *The expected number of events processed by a kinetic hanger sweeping an arrangement of m δ-intersecting curve segments such that any vertical line intersects at most n segments is $O(m/n\lambda_{\delta+2}(n) \lg n)$. An extra $O(\lg n)$ time must be spent processing each event.*

PROOF. Divide the arrangement into $\Theta(m/n)$ vertical slabs with $\Theta(n)$ segments in each slab. □

Corollary 10 *The expected number of events processed by a kinetic hanger sweeping an arrangement of m δ-intersecting curves such that any vertical line intersects at most n curves is $O(m/n\lambda_{\delta}(n) \lg n)$. An extra $O(logn)$ time must be spent processing each event.*

3 Conclusion

In this paper, we introduced a new randomized kinetic priority queue, the kinetic hanger. The kinetic hanger has the same expected time complexities of the kinetic tournament (deterministic) and the kinetic heater, while possessing some advantages compared to these structures. The fact that the kinetic hanger is very simple to implement and has $O(1)$ locality makes it a strong candidate for practical implementation.

When compared to the kinetic tournament, the kinetic hanger has $O(1)$ locality, instead of the tournament's $O(\lg n)$ locality. When compared to the kinetic heater, the kinetic hanger is faster in practice and easier to implement because it does not use rotation. The kinetic hanger is also faster in practice because L_n, the level of the n-th largest element, satisfies $E(L_n) \leq \lg n$, while, in the

kinetic heater, it satisfies $E(L_n) \simeq 2 \ln n \simeq 1.39 \lg n$. When compared to the kinetic heap, the kinetic hanger seems to be a randomized version of it, for which we could find tight and efficient complexity bounds, while maintaining the simplicity of the heap.

Tight analysis for the maximum number of events in the kinetic heap remains an intriguing question. Using arguments similar to the ones we used, one can show that a "random kinetic heap" is efficient in expectation. By "random kinetic heap" we mean a kinetic heap initially constructed with a random permutation of the set of elements. Unfortunately, insertions and deletions destroy the randomness of this structure. Determining whether a kinetic heap is efficient in the worst case, or subject to insertions and deletions remains unsolved.

Variations of the kinetic heap and the kinetic tournament have been proposed in [9] and [6], respectively, where the tree is a $(\lg n)$-ary tree, instead of a binary tree. These variations reduce the time complexities of the structures by a factor of $\Theta(\lg \lg n)$. The binary search tree structure of the kinetic heater makes it hard to use the same strategy, but a k-ary version of the kinetic hanger is simple and natural. Instead of using random bits, we use random numbers with k possible values.

The time complexity of processing an event in a k-ary hanger with n elements is $O(k + \lg(n/k))$. If $k = O(\lg n)$, the time complexity of processing an event remains $O(\lg n)$.

We conjecture that the expected number of events processed by a kinetic k-ary hanger sweeping an arrangement Λ of n δ-intersecting curve segments is $O(\lambda_{\delta+2}(n) \lg n / \lg k)$, and that the product of the number of events and the time to process each event is minimum for $k = O(\lg n)$.

Acknowledgements We are grateful to the referee for his careful reading and many suggestions which helped to improve the paper. We thank Angelika Steger for supplying a simple proof of Lemma 6. The first author is supported by Coordenação de Aperfeiçoamento de Pessoal de nível Superior - CAPES.

References

[1] C. R. Aragon and R. G. Seidel. Randomized search trees. *Algorithmica*, 16(4/5):464–497, 1996.

[2] J. Basch, L. J. Guibas, and G. D. Ramkumar. Reporting red-blue intersections between two sets of connected line segments. In *Proceedings of the 4th Annual European Symposyum on Algorithms*, volume 1136 of *Lecture Notes Comput. Sci.*, pages 302–319. Springer-Verlag, 1996.

[3] J. Basch, L. J. Guibas, and G. D. Ramkumar. Sweeping lines and line segments with a heap. In *Proceedings of the 13th Annual Symposyum on Computational Geometry*, pages 469–471, 1997.

[4] Julien Basch, Leonidas J. Guibas, and John Hershberger. Data structures for mobile data. *Journal of Algorithms*, 31(1):1–28, April 1999.

[5] G. S. Brodal and R. Jacob. Dynamic planar convex hull with optimal query time. In *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory, SWAT2000*, volume 1851 of *Lecture Notes Comput. Sci.*, pages 57–70. Springer-Verlag, 2000.

[6] T. M. Chan. Remarks on k-level algorithms in the plane. http://www.math.uwaterloo.ca/~tmchan/lev2d_7_7_99.ps.gz, 1999.

[7] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete and Computational Geometry*, 4:387–421, 1989.

[8] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1990.

[9] G. D. da Fonseca and C. M. H. de Figueiredo. Kinetic heap-ordered trees: tight analysis and improved algorithms. *Information Processing Letters*, 85:165–169, 2003.

[10] R. Graham, D. Knuth, and O. Patashnik. *Concrete Mathematics - A Foundation for Computer Science*. Addison-Wesley Publishing Company, 1994.

[11] M. Sharir and P. K. Agarwal. *Davenport-Schinzel Sequences and Their Geometric Applications*. Cambridge University Press, 1995.