Thesis Proposal: Multi-Splay Trees

Chengwen Chris Wang

February 24, 2006

Abstract

In this thesis, we introduce multi-splay trees (MSTs) and prove several results demonstrating that MSTs have most of the useful properties different BSTs have. First, we prove a close variant of the splay tree access lemma [ST85] for multi-splay trees, a lemma that implies MSTs have the $O(\log n)$ runtime property, the static finger property, and the static optimality property. Then, we extend the access lemma by proving the remassing lemma, which is similar to the reweighting lemma for splay trees [Geo04]. The remassing lemma shows that MSTs satisfy the working set property and key-independent optimality and are competitive to parametrically balanced trees, as defined in [Geo04]. After proving the remassing lemma, we also prove that MSTs achieve the $O(\log \log n)$-competitiveness and that sequential access in MSTs costs $O(n)$.

Then we extend the static model naturally to allow insertions and deletions and show how to carry out these operations in multi-splay trees to achieve $O(\log \log n)$-competitiveness, a result no other BST scheme has been proved to have. In addition, we prove that MSTs satisfy the deque property, which is still an open problem for splay trees since it was conjectured in 1985 [Tar85]. While it is easy to construct a BST that trivially satisfies the deque property, no other BST scheme satisfying other useful properties has been proved to have deque property. Together, these results show that MSTs satisfy most of the important properties satisfied by different binary search trees.

1 Introduction and Motivation

Efficiently maintaining and manipulating totally ordered sets is a fundamental problem in computer science. The sets manipulated by algorithms can grow, shrink and change over time. Specifically, many algorithms need a data structure that can efficiently support several operations, such as insertion, deletion, predecessor, successor and membership testing. A standard data structure that maintains a totally ordered set and supports these operations is a binary search tree (BST). Various binary search trees have been independently invented by a number of researchers in the early 1960s [Knu73]. Over the years, many binary search trees achieved the theoretical minimum number of $O(\log n)$ key comparisons needed per operation. Hence, many BST algorithms are optimal (up to a constant) using worst case analysis. However, for many

sequences σ of m operations, the optimal cost for executing the sequences is $o(m \log n)$. To exploit the patterns of query sequences from specific applications, such as randomly and independently drawn queries from a fixed distribution[1], finger search[2], and sequential queries[3], researchers have designed specialized binary search trees that efficiently support various types of patterns.

In 1985, Sleator and Tarjan [ST85, Tar85] showed that it is possible to efficiently handle all the query patterns mentioned above (and many more) in a single binary search tree called *splay tree*. A splay tree is a self-adjusting form of binary search tree such that each time a node in the tree is accessed, that node is moved to the root according to an algorithm called *splaying*. In a splay tree, all accesses and updates (e.g. query, insert, delete) are accomplished by using the splaying algorithm. A splay tree has a number of remarkable properties explained in Section 4, including the Balance Theorem [ST85], the Static Optimality Theorem [ST85], the Static Finger Theorem [ST85], the Working Set Theorem [ST85], the Scanning Theorem [Sun89a, Tar85, Sun92, Elm04], the Reweighting Lemma, [Geo04] the Dynamic Finger Theorem [CMSS00, Col00], the Key Independence properties [Iac02], and competitiveness to parametrically balanced trees [Geo04]. Since splay tree has satisfied many necessary properties of an $O(1)$-competitive binary search tree, Sleator and Tarjan proposed the Dynamic Optimality Conjecture, which states splay tree (without insertions and deletions) is $O(1)$-competitive to the optimal off-line binary search tree. After more than 20 years, the Dynamic Optimality Conjecture remains an open problem.

Since no one has shown that any binary search tree is $O(1)$-competitive, Demaine, Harmon, Iacono and Pătraşcu [DHIP04] suggested searching for alternative binary search tree algorithms that have small but non-constant competitive factors. They proposed *tango*, a BST algorithm that achieves a competitive ratio of $O(\log \log n)$. Tango is the first data structure proved to achieve a nontrivial competitive factor. Unfortunately, tango does not satisfy many of the necessary conditions of a constant competitive binary search tree.

In this thesis, we introduce a new data structure called multi-splay trees. We prove that multi-splay trees (MST's) can efficiently execute most query sequences proven to execute efficiently on most binary search trees.

2 Binary Search Tree Model

In order to discuss the optimality of BST algorithms, we need to give a precise definition of this class of algorithms and their costs. The model we use is implied by Sleator and Tarjan [ST85] and developed in detail by Wilber [Wil89]. A static set of n keys is stored in the nodes of a binary tree. The keys are from a totally ordered universe, and they are stored in symmetric (left to right) order. Each node has a pointer to its left child, to its right child, and to its parent. Also, each node may keep $o(\log n)$ bits of additional information but no additional pointers.

[1]See [Knu71, Fre75, Meh75, Meh79, GW77, HT71, HKT79, Unt79, Hu82, Kor81, KV81, BST85]

[2]See [BY76, GMPR77, Tsa86, TvW88, HL79, Har80, HM82, Fle93, SA96, BLM+03, Pug89, Pug90, Iac01b, Bro05]

[3]See [Tar85, Sun89a, Sun92, Elm04]

A BST algorithm is required to process a sequence of queries $\sigma = \sigma_1, \sigma_2, \ldots, \sigma_m$. Each access σ_i is a query to a key $\hat{\sigma_i}$ in the tree[4], and the requested nodes must be accessed in the specified order. Each access starts from the root and follows pointers until the desired node (the one with key $\hat{\sigma_i}$) is reached. The algorithm is allowed to update the fields in any node or rotate any edges that it touches along the way. The cost of the algorithm to execute a query sequence is defined to be the number of nodes touched.

Finally, we do not allow any information to be preserved from one access to the next, other than the nodes' fields and a pointer to the root of the tree. It is easy to see that this definition is satisfied by all of the standard BST algorithms, such as red-black trees and splay trees.

3 Competitive Analysis on BST

Given any initial tree T_0 and any m-element access sequence σ, for any BST algorithm satisfying these requests, the cost can be defined using the model in Section 2. Thus, we can define $\mathrm{OPT}(T_0, \sigma)$ to be the minimum cost of any BST algorithm for satisfying these requests starting with initial tree T_0. Furthermore, since the number of rotations needed to change any binary search tree of n nodes into another one is at most $O(n)$ [Cra72, CW82, STT86, Mäk88, LP89], it follows that $\mathrm{OPT}(T_0, \sigma)$ differs from $\mathrm{OPT}(T_0', \sigma)$ by at most $O(n)$. Thus, as long as $m = \Omega(n)$, the initial tree is irrelevant. We denote the off-line optimal cost starting from the *best* possible initial tree as $\mathrm{OPT}(\sigma)$. Similarly, for any on-line binary search algorithm A, $A(\sigma)$ denotes the on-line cost to execute σ starting from the *worst* initial tree. Because the initial tree of a BST algorithm could be a very unbalanced binary search tree, we assume the number of operations, m, is greater than $n \log n$ to avoid unfairly penalizing the on-line BST algorithm.

An on-line binary search tree algorithm A is T-competitive if

$$\forall \sigma A(\sigma) < T * \mathrm{OPT}(\sigma) + O(m).$$

Currently, the best competitive on-line binary search trees [DHIP04, SW04, Geo05, WDS06] are $O(\log \log n)$-competitive. While we do not know if it is possible to have an $O(1)$-competitive BST, we know an extensive list of properties that any $O(1)$-competitive BST must satisfy.

4 Properties of an $O(1)$-competitive BST

Before we move on to discuss the properties identified as necessary for an $O(1)$-competitive BST, let us first discuss the assumptions of this section. In this section, all sequences of operations are assumed to involve only queries. We call a sequence without insertions and deletions a *query sequence*. Since the set of keys do not change, we can assume WLOG that there are n keys numbered from $1, 2, \ldots, n$.

Now we are ready for a complete list of the useful binary search tree properties.

[4]WLOG, this model is only concerned with successful searches [AW98].

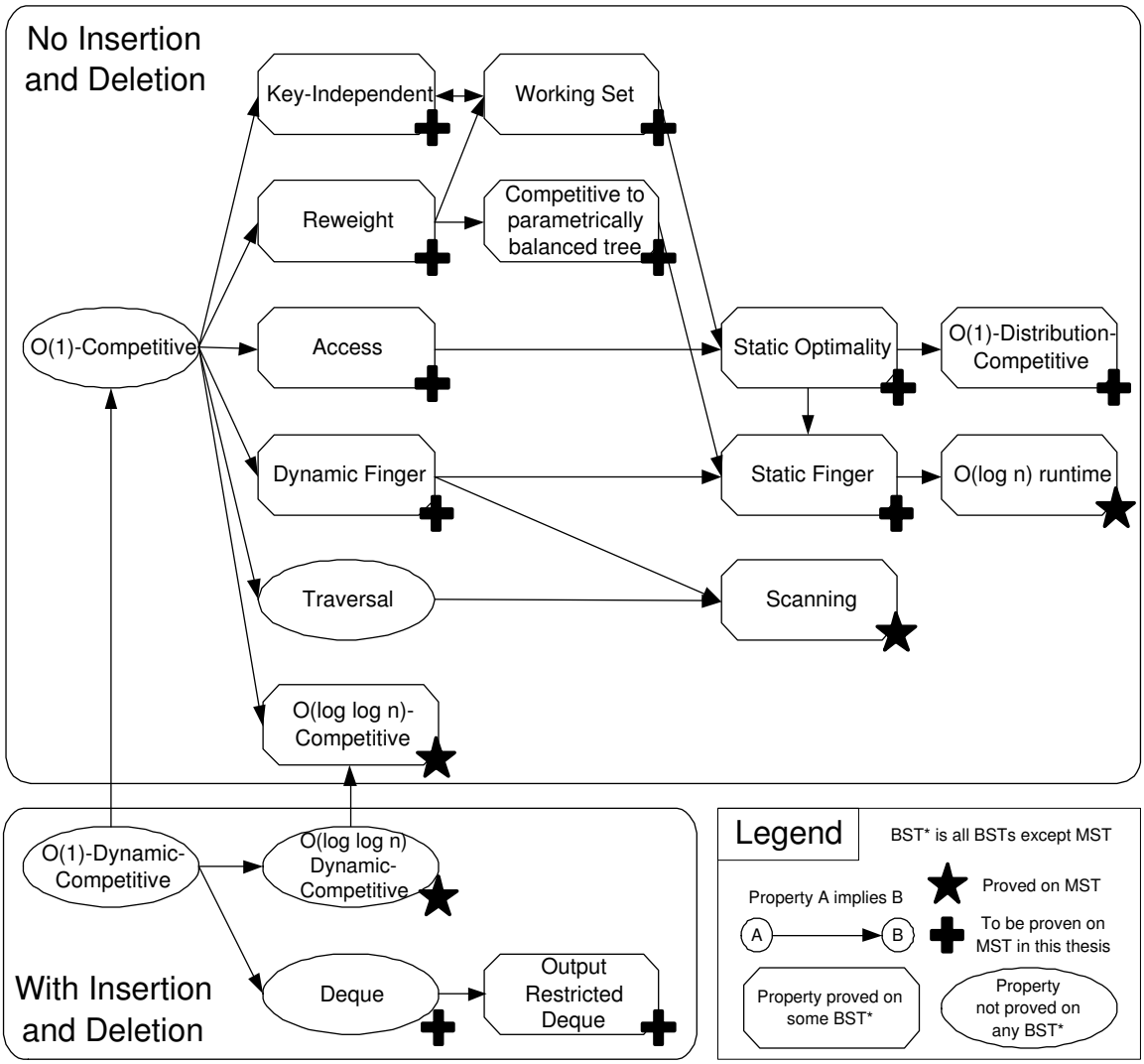

Figure 1: This figure shows the implication relationships for the list of properties in Section 4 and Section 6. The minimum set of edges are shown so that the transitive closure of the above graph includes all the implications. This graph contains almost all of the analyzed static query patterns. I hypothesize that multi-splay trees efficiently execute most classes of query patterns proved on BSTs plus a few more. I hope to prove that MSTs have all of the above properties in my thesis except the $O(1)$-competitive property, the $O(1)$-dynamic-competitive property, and the traversal property. (Maverick Woo conjectured that MSTs have the dynamic finger property [private communication]. If Maverick Woo proves the dynamic finger property on MSTs, then the proof will not be part of my thesis.) If I can prove my hypothesis, then multi-splay trees would be the only data structure proved to satisfy both the $O(\log \log n)$-dynamic competitive property and the deque property.

Property 1. A binary search tree structure has the $O(\log n)$ *runtime* property if it executes every σ in time $O(m \log n)$

In the worst case, some query sequences will need $\Omega(m \log n)$ time [Wil89]. Thus, having this property implies the data structure is theoretically optimal under worst case analysis. Almost every binary search tree has the $O(\log n)$ runtime property.

Property 2. [ST85] A binary search tree structure has the *static finger* property if it executes every σ in time $O(m + \sum_{i=1}^{m} log(|f - \sigma_i| + 1))$ for all possible finger f.

There exists a specialized data [GMPR77, Bro98] structure for a fix initialization of f. However, for a data structure to have the static finger property, it must have the finger search running time for all possible fingers f without knowing the finger at initialization.

Property 3. [AW98] A binary search tree structure A is $O(1)$-*distribution-competitive* if for all n, all distributions D on n elements and all initial trees T_0, the expected cost for A to serve a request is less than a constant times the optimal static tree for distribution D.

An example of a binary search tree that satisfies *only* the $O(1)$-distribution-competitive property is the *move-to-root* binary search tree [AM78]. This binary search tree always rotates the queried node x repeatedly until x become the root. Because the optimal static tree for a fixed distribution D is a static tree, the $O(1)$-distribution-competitive property is implied by the *static optimality* property.

Property 4. [ST85] A binary search tree structure has the *static optimality* property if the time to execute σ is $O(m + \sum_{i=1}^{m} f(i) \log(m/f(i))$, where $f(i)$ is the number of times key i is queried.

Because $\Omega(\sum_{i=1}^{m} f(i) \log(m/f(i)))$ [Abr63] is an informational theoretical lower bound on the worse sequences of queries with frequency $f(i)$, the binary search trees with static optimality is constant competitive to any static binary search tree, including the optimal static tree for distribution D. Several data structures [Knu71, Fre75, Meh75, Meh79, GW77, HT71, HKT79, Unt79, Hu82, Kor81, KV81, BST85] have the static optimality, but they need to know $m/f(i)$ during initialization. On the other hand, Splay tree has the static optimality property without knowing the frequency $m/f(i)$ in advance. Any data structure with the static optimality property also has the static finger property [Iac01a].

Property 5. [ST85] A binary search tree structure has the *working set* property if the time to execute σ is $O(m + \sum_{i=1}^{m} \log d(l(i), i))$, where $d(i, j)$ is the number of distinct keys accessed in the subsequence $\sigma_i, \sigma_{i+1} \ldots \sigma_j$, and $l(i)$ is the index of the last access to σ_i in the subsequence $\sigma_1, \sigma_2, \ldots \sigma_{i-1}$. ($l(i) = d(1, i-1)$ if σ_i does not appear in the subsequence.)

The working set property implies both static finger and static optimality. It also implies that if all queries are in a small subset of keys of size k, then the query sequence can be executed in $O(m \log k)$. In many applications, such as compression [Jon88, GRVW95], a recently queried element is likely to be queried again. These recent queries are exactly the element with low amortized cost in the working set property.

Property 6. [Iac02] A binary search tree structure has the *key-independent* property if the time to execute σ is $O(E[OPT(b(\sigma))])$, where b is random bijection of the keys from n to n.

Iacono [Iac02] introduced the key-independent optimality as another necessary condition for an $O(1)$-competitive binary search tree, and he proved that the key independent property is equivalent to the working set property up to a multiplicative constant factor.

Property 7. [ST85] A binary search tree structure has the *access* property if for any positive weight assignment $w(x)$ for each element, the time to execute σ is $O(m + \sum_{i=1}^{m} \log \frac{W}{s(\sigma_i)})$, where $W = \sum_{i=1}^{n} w(i)$, $s(\sigma_i)$ must be greater than $w(\sigma_i)$ (and $s(x)$ can depend on the structure of the binary search tree).

With a simple weight assignment, this property implies the static finger and the static optimality properties [ST85]. Because of the flexibility in assigning weights, the access property can be used to combine and generalize properties proved with weight assignment. For instance, the access property implies that for any constant number of finger $f_1, f_2 ... f_k$, the amortized cost to execute a sequence is summation of the log of the distance to the closest finger. That is, $O(m + \sum_{i=1}^{m} \min_j \log(|f_j - a_i| + 1))$.

Property 8. [Geo04] Let $w_i(x)$ be any positive weight assignment of x right before i^{th} query. A binary search tree structure has the *reweight* property if the time to execute σ is $O(m + \sum_{i=1}^{m} \log \frac{W_i}{w_i(\sigma_i)} + \sum_{i=2}^{m} \sum_{j=1}^{n} \log \max(0, \frac{w_i(j)}{w_{i-1}(j)}))$, where $W_i = \sum_{j=1}^{n} w_i(j)$.

This property is almost the same as the access property with an additional reweight operation, and the cost to increase the weight of element from *old* to *new* is roughly $O(\log \frac{new}{old})$. The reweight operation is not an operation in the data structure, it is merely used in the analysis. The reweight operation enables the analysis to adapt to the query patterns and prove tighter bounds [Geo04].

Property 9. A binary search tree structure has the *dynamic finger* property if the time to execute σ is $O(m + \sum_{i=2}^{m} \log(d(i) + 1))$, where $d(i)$ is the difference in rank between the ith query and the $(i-1)$th query.

Brodal [Bro05] wrote a chapter on finger search trees and some of the common data structures with the dynamic finger property. Several search trees [BY76, GMPR77, Tsa86, TvW88, HL79, Har80, HM82, Fle93, SA96, BLM+03] have this property, but many violate the definition of Binary Search Tree. For instance, the level linked (2, 4)-tree of Huddleston and Mehlhorn [HM82] and unified data structure of Iacono [Iac01b, BD04] use extra pointers that are not valid in the BST model; randomized skip lists of Pugh [Pug89, Pug90] duplicates the same key multiple times, which is a violation of the BST model; or the auxiliary *hand* data structure of Blelloch, Maggs and Woo [BMW03] maintains extra pointers into a degree balanced binary search tree. Splay tree [ST85] is one of the few data structure that satisfies the binary search tree model and has the dynamic finger property [CMSS00, Col00].

Property 10. [Tar85, ST85] A binary search tree structure has the *scanning* property if the time to execute $\sigma = 1, 2, 3, \ldots, n$ is $O(n)$ starting at any valid initial tree.

Property 11. [ST85] A binary search tree data structure has the *traversal* property if given any initial tree T_0 and a input tree T_i, the cost of sequentially querying elements in the order they appear in preorder of T_i is $O(n)$.

When T_i is a right path, the elements in preorder of T_i is $1, 2, \ldots n$, which is exactly the query sequence in the scanning property. Thus, any data structure that satisfies this property also satisfies the scanning property. While any $O(1)$-competitive binary search tree must have the traversal property, no binary search tree is known to have the traversal property. However, special case of the traversal property (when $T_0 = T_i$ [CH93], or when T_i is a right path [Sun89a, Tar85, Sun92, Elm04]) was proved for splay trees.

Property 12. A binary search tree data structure has the $O(\log\log n)$-*competitive* property if it executes σ in time $O(\log\log n) * OPT(\sigma)$.

The $O(\log\log n)$-competitive property is currently the best competitive (and only non-trivial) bound proved for a binary search tree [DHIP04, SW04, Geo05, WDS06].

Property 13. A binary search tree structure is *competitive to parametrically balanced trees* if the data structure is $O(1)$-competitive to parametrically balanced trees.

Parametrically balanced trees is a large class of balance search trees introduced by Georgakopoulos [Geo04]. The class includes most balanced trees, such as BB(α)-trees [NR73, BM80], AVL-trees [AVL62], half-balanced trees [Oli82, Ove83], B-tree [RB72]. These parametrically balanced trees are allowed to restructure based on future queries and pay a small cost proportional to the number of local changes in the structure. Since Georgakopoulos [Geo04] has a detailed description on this class of balanced trees, we omit the details here.

To continue with properties that include insertion and deletion, we must first define the dynamic BST model.

5 Dynamic BST model

Before we can reason about the properties and competitiveness of dynamic binary search trees, we must introduce an intuitive definition of what it means for a dynamic BST to be competitive. We assume an arbitrary dynamic BST algorithm A must execute a sequence of operations $\sigma = \sigma_1, \ldots, \sigma_m$, each of which is *query*($\hat{\sigma}_i$), *insert*($\hat{\sigma}_i$), or *delete*($\hat{\sigma}_i$). For each σ_i, we assume A must pay the following costs:

- To execute *query*($\hat{\sigma}_i$), it must pay for touching each node on the path from the root to $\hat{\sigma}_i$.
- To execute *insert*($\hat{\sigma}_i$), it must pay for inserting the node at a leaf in addition to the traversal to get there. This is reasonable because A must search for $\hat{\sigma}_i$ to realize its BST does not contain $\hat{\sigma}_i$.

- To execute *delete*($\hat{\sigma}_i$), it must pay for accessing $\hat{\sigma}_i$ and for performing rotations until $\hat{\sigma}_i$ has no children (at which point, the node can be removed).[5]

During (or after) each operation, a BST algorithm may perform any rotations it wishes at the cost of one per rotation. The cost of an operation is simply the total number of nodes touched, plus the number of rotations. Without insertions and deletions, this definition would be identical to the one in Section 2. We use DOPT(σ) to refer to the cost of an optimal off-line dynamic BST algorithm executing σ (starting from the *best* possible initial tree).

6 Properties of an $O(1)$-dynamic-competitive BST

Partly because competitive analysis in standard BST model is already difficult, there are few results on the dynamic BST model. Below is a list of properties researchers have considered or mentioned. Of the properties below, the $O(\log\log n)$-dynamic-competitive property and the deque property is not proved on any other binary search tree. In my thesis, I plan to prove that MSTs satisfy all of the following properties.

Property 14. [Tar85] A binary search tree has the *deque* property if a sequence of m push, pop, inject and eject operation is executed in time $O(m+n)$.

Splay trees are conjectured to satisfy the deque property [Tar85]. Lucas [Luc88] showed that the total cost of a sequence of ejects and pops is $O(n\alpha(n,n))$ if the initial tree is a simple path of n nodes. Currently, the best bound is proved by Sundar [Sun89a, Sun89b, Sun92]. He showed that splay trees can execute a sequence of m deque operations on n nodes in $O((m+n)\alpha(m+n,n+n))$.

Tarjan [Tar85] proved that splay tree satisfy a special case of the deque property - the output restricted deque property.

Property 15. A binary search tree has the *output restricted deque* property if a sequence of m push, pop and inject operation is execute in time $O(m+n)$

Property 16. A binary search tree has the $O(\log\log n)$-*dynamic-competitive* property if it execute every sequence σ of queries, inserts and deletes in time $O(\log\log n * DOPT(\sigma))$.

MST is the only data structure proved to satisfy this property. However, it may be possible to prove this property on many of the existing $O(\log\log n)$-competitive BSTs with some small modifications.

7 Previous MST related work and future plans

Wang, Derryberry and Sleator [WDS06] showed that MSTs satisfy the $O(\log n)$ run-time property, the scanning property, the $O(\log\log n)$-competitive property and the

[5] In this model, we do not allow BSTs to swap nodes and contract edges during deletion. This implies that it must also pay for accessing both *pred*($\hat{\sigma}_i$) and *succ*($\hat{\sigma}_i$). Because of this restriction, small modifications are necessary to include many standard binary search trees in this model.

$O(\log \log n)$-dynamic-competitive property. From private conversation with Maverick Woo, MSTs are also likely to have the dynamic finger property (work in progress).

I hypothesize that multi-splay trees efficiently execute most classes of query patterns proved on BSTs plus a few more. I hope to prove in this thesis that MSTs satisfy all of the properties in Figure 1 except the $O(1)$-competitive property, the $O(1)$-dynamic-competitive property and the traversal property. Excluding those that are already proved, I expect to finish the proofs for the following properties (in alphabetical order): the access property, the competitive to parametrically balanced tree property, the deque property, the key-independent property, the $O(1)$ distribution-competitive property, the output restricted deque property, the reweight property, the static finger property, the static optimality property, and the working set property. In fact, if I can prove my hypothesis, then multi-splay tree would be the only binary search tree proved to satisfy both the $O(\log \log n)$-dynamic competitive property and the deque property, which was conjectured on splay trees in 1985. (While it is easy to construct a BST that trivially satisfies the deque property, no other BST scheme satisfying other useful properties has been proved to have deque property.)

8 Acknowledgment

I am pleased to acknowledge the people who made this research possible. First and foremost, I am deeply indebted to Professor Daniel Sleator for his unwavering support and valuable suggestions. I am grateful to Professor Gary Miller for many useful suggestions and helpful discussions. I owe a debt of gratitude to Jonathan Derryberry for extensive discussions that stimulated my thinking and helped to clarify many of the ideas presented. Lastly, many thanks to Maverick Woo for his helpful suggestions and constructive criticisms.

References

[Abr63] N. Abramson. *Information Theory and Coding.* McGraw-Hill, New York, 1963.

[AM78] Brian Allen and Ian Munro. Self-organizing binary search trees. *J. ACM*, 25(4):526–535, 1978.

[AVL62] G.M. Adel'son-Vel'ski and E.M. Landis. An algorithm for the organization of information. *Soviet Math. Dokl.*, 3:1259–1263, 1962.

[AW98] Susanne Albers and Jeffery Westbrook. Self-organizing data structures. In *Developments from a June 1996 seminar on Online algorithms*, pages 13–51, London, UK, 1998. Springer-Verlag.

[BD04] Mihai Bădoiu and Erik D. Demaine. A simplified and dynamic unified structure. In *Proceedings of the 6th Latin American Symposium on Theoretical Informatics (LATIN 2004)*, volume 2976 of *Lecture Notes in Computer Science*, pages 466–473, Buenos Aires, Argentina, April 5–8 2004.

[BLM+03] Gerth S. Brodal, George Lagogiannis, Christos Makris, Athanasios Tsakalidis, and Kostas Tsichlas. Optimal finger search trees in the pointer machine. *Journal of Computer and System Sciences, Special issue on STOC 2002*, 67(2):381–418, 2003.

[BM80] N. Blum and K. Mehlhorn. On the average number of rebalancing operations in weight balanced trees. *Theoretical Computer Science*, 11:303–320, 1980.

[BMW03] G. Blelloch, B. Maggs, and M. Woo. Space-efficient finger search on degree-balanced search trees, 2003.

[Bro98] Gerth S. Brodal. Finger search trees with constant insertion time. In *SODA '98: Proceedings of the ninth annual ACM-SIAM symposium on Discrete algorithms*, pages 540–549, Philadelphia, PA, USA, 1998. Society for Industrial and Applied Mathematics.

[Bro05] Gerth S. Brodal. Finger search trees. In Dinesh Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, chapter 11, page 11. CRC Press, 2005.

[BST85] S. Bent, D. Sleator, and R. Tarjan. Biased search trees. *SIAM Journal of Computing*, 14:545–568, 1985.

[BY76] J.L. Bently and A. C.-C. Yao. An almost optimal algorithm for unbounded searching. *Information Processing Letters*, 5(3):82–87, 1976.

[CH93] R. Chaudhuri and H. Höft. Splaying a search tree in preorder takes linear time. *SIGACT News*, 24(2):88–93, 1993.

[CMSS00] Richard Cole, Bud Mishra, Jeanette Schmidt, and Alan Siegel. On the dynamic finger conjecture for splay trees. Part I: Splay Sorting log n-Block Sequences. *Siam J. Comput.*, 30:1–43, 2000.

[Col00] Richard Cole. On the dynamic finger conjecture for splay trees. Part II: The Proof. *Siam J. Comput.*, 30:44–85, 2000.

[Cra72] C.A. Crane. Linear lists and priority queues as balanced binary trees. Technical Report STAN-CS-72-259, Dept. of COmputer Science, Stanford University, 1972.

[CW82] K. Culik, II and D. Wood. A note on some tree similarity measures. *Inform. Process. Lett.*, pages 39–42, 1982.

[DHIP04] Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Pătraşcu. Dynamic Optimality–Almost. *FOCS*, 2004.

[Elm04] Amr Elmasry. On the sequential access theorem and deque conjecture for splay trees. *Theoretical Computer Science*, 314:459–466, 2004.

[Fle93] Rudolf Fleischer. A simple balanced search tree with o(1) worst-case update time. In *ISAAC '93: Proceedings of the 4th International Symposium on Algorithms and Computation*, pages 138–146, London, UK, 1993. Springer-Verlag.

[Fre75] M. L. Fredman. Two applications of a probabilistic search technique: sorting x + y and building balanced search trees. *Proc. Seventh ACM symposium on Theory of Computing*, pages 240–244, 1975.

[Geo04] George F. Georgakopoulos. Splay trees: a reweighing lemma and a proof of competitiveness vs. dynamic balanced trees. *Journal of Algorithms*, 51(1):64–76, April 2004.

[Geo05] George F. Georgakopoulos. How to splay for loglogn-competitiveness. In Sotiris E. Nikoletseas, editor, *Experimental and Efficient Algorithms: 4th International Workshop, WEA 2005*, 2005.

[GMPR77] Leo J. Guibas, Edward M. McCreight, Michael F. Plass, and Janet R. Roberts. A new representation for linear lists. In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 49–60, New York, NY, USA, 1977. ACM Press.

[GRVW95] Dennis Grinberg, Sivaramakrishnan Rajagopalan, Ramarathnam Venkatesan, and Victor K. Wei. Splay trees for data compression. In *SODA '95: Proceedings of the sixth annual ACM-SIAM symposium on Discrete algorithms*, pages 522–530, Philadelphia, PA, USA, 1995. Society for Industrial and Applied Mathematics.

[GW77] A. M. Garsia and M. L. Wachs. A new algorithm for minimal binary search trees. *SIAM Journal on Computing*, 6:622–642, 1977.

[Har80] D. Harel. Fast updates of balanced search trees with a guaranteed time bound per update. Technical Report 154, University of California, Irvine, 1980.

[HKT79] T. C. Hu, D. J. Kleitman, and J. K. Tamaki. Binary search trees optimum under various criteria. *SIAM J. Appl. Math.*, 37:246–256, 1979.

[HL79] D. Harel and G.S. Lueker. A data structure with movable fingers and deletions. Technical Report 145, University of California, Irvine, 1979.

[HM82] S. Huddleston and K. Mehlhorm. A new data structure fore representing sorted lists. *Acta Informatica*, 17:157–184, 1982.

[HT71] T. C. Hu and A. C. Tucker. Optimal computer-search trees and variable-length alphabetic codes. *SIAM J. Appl. Math.*, 21:514–532, 1971.

[Hu82] T. C. Hu. *Combinatorial Algorithms.* Addison-Wesley, Reading, MA, 1982.

[Iac01a] J. Iacono. *Distribution Sensitive Data Structures.* PhD thesis, Rutgers, The State University of New Jersey, Graduate School, New Brunswick, 2001.

[Iac01b] John Iacono. Alternatives to splay trees with o(log n) worst-case access times. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 516–522, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[Iac02] John Iacono. Key independent optimality. In *ISAAC '02: Proceedings of the 13th International Symposium on Algorithms and Computation*, pages 25–31, London, UK, 2002. Springer-Verlag.

[Jon88] D. W. Jones. Application of splay trees to data compression. *Commun. ACM*, 31(8):996–1007, 1988.

[Knu71] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

[Knu73] D. E. Knuth. *The art of computer programming, vol. 3: Sorting and searching*. Addison-Wesley, Reading, MA, 1973.

[Kor81] J. F. Korsch. Greedy binary search trees are nearly optimal. *Inform. Proc. Letters*, 13:16–19, 1981.

[KV81] H. P. Kriegel and V. K. Vaishnavi. Weighted multidimensional b-trees used as nearly optimal dynamic dictionaries. *Mathematical Foundations of Computer Science*, 1981.

[LP89] F. Luccio and L. Palgi. On the upper bound on the rotation distance of binary trees. *Inf. Process. Lett.*, 31(2):57–60, 1989.

[Luc88] J. M. Lucas. Arbitrary splitting in splay trees. Technical Report DCS-TR-234, Rutgers University, 1988.

[Mäk88] Erkki Mäkinen. On the rotation distance of binary trees. *Inf. Process. Lett.*, 26(5):271–272, 1988.

[Meh75] K. Mehlhorn. Nearly optimal binary search trees. *Acta Inform.*, 5:287–295, 1975.

[Meh79] K. Mehlhorn. Dynamic binary search. *SIAM Journal on Computing*, 8:175–198, 1979.

[NR73] J. Nievergelt and E.M. Reingold. Binary search trees of bounded balanced. *SIAM J. on Computing*, 2:33–43, 1973.

[Oli82] H. J. Olivié. A new class of balanced search trees: half balanced search trees. *Theoret. Inform. Appl.*, 16:51–71, 1982.

[Ove83] M. H. Overmars. The design of dynamic data structures. In *Lecture Notes in Computer Science*, volume 156. Springer-Verlag, Heidelberg, 1983.

[Pug89] W. Pugh. A skip list cookbook. Technical Report CS-TR-2286.1, Dept. of Computer Science, University of Maryland, 1989.

[Pug90] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.

[RB72] E. McCreight R. Bayer. Organization and maintenance of large ordered indexes. *Acta Informatica*, 1:173–189, 1972.

[SA96] Raimund Seidel and Cecilia R. Aragon. Randomized search trees. *Algorithmica*, 16(4/5):464–497, 1996.

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, July 1985.

[STT86] D. D. Sleator, R. E. Tarjan, and W. P. Thurston. Rotation distance, triangulations, and hyperbolic geometry. *Proc. 18th Annual ACM Symposium on Theory of Computing*, pages 122–135, 1986.

[Sun89a] R. Sundar. Twists, turns, cascades, deque conjecture, and scanning theorem. *Proceedings of the 13th Symposium on Foundations of Computer Science*, pages 555–559, 1989.

[Sun89b] R. Sundar. Twists, turns, cascades, deque conjecture, and scanning theorem. Technical Report 427, Courant Institue, New York University, 1989.

[Sun92] R. Sundar. On the deque conjecture for the splay algorithm. *Combinatorica*, 12:95–124, 1992.

[SW04] D. D. Sleator and C. C. Wang. Dynamic optimality and multi-splay trees. Technical Report CMU-CS-04-171, Carnegie Mellon University, 2004.

[Tar85] R. Tarjan. Sequential access in splay trees takes linear time. *Combinatorica*, 5:367–378, 1985.

[Tsa86] Athanasios K Tsakalidis. Avl-trees for localized search. *Inf. Control*, 67(1-3):173–194, 1986.

[TvW88] Robert E. Tarjan and Christopher van Wyk. An o (n log log n)-time algorithm for triangulating a simple polygon. *SIAM J. Comput.*, 17(1):143–178, 1988.

[Unt79] K. Unterauer. Dynamic weighted binary search trees. *Acta Inform.*, 11:341–362, 1979.

[WDS06] Chengwen Wang, Jonathan Derryberry, and Daniel D. Sleator. $O(\log \log n)$-Competitive Dynamic Binary Search Tree. *SODA*, 2006.

[Wil89] Robert Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM Journal of Computing*, 18(1):56–67, 1989.