Competitive Tree-Structured Dictionaries

MICHAEL T. GOODRICH*

Abstract

In this note we describe a general technique for making tree-structured dynamic dictionaries adapt to be competitive with the most efficient implementation, by using *potential energy* parameters and a simple partial rebuilding scheme.

Introduction. On-line algorithms deal with optimizing the performance of operation sequences (e.g., see [4, 9]). Such algorithms are desired to be *c-competitive* [4], for some parameter $c > 0$, where c is an upper bound on the ratio of the costs of the solution defined by the on-line algorithm and an oracle's algorithm. In this paper we are interested in dictionary data structures that are competitive in this same sense.

Our Results. We present a simple adaptive tree-based dictionary structure that is balanced and competitive. Our approach is based on a *potential energy* parameter stored at each node in the tree. As updates and queries are performed, the potential energy of tree nodes are increased or decreased. Whenever the potential energy of a node reaches a *threshold* level, we rebuild the subtree rooted at that node. We show that, in spite of its conceptual simplicity, such a scheme is constant-ratio competitive with a static oracle using *a priori* knowledge of the operation distribution.

Related Prior Work. Besides general work for on-line algorithms (e.g., see [4, 9]) and data structures that use partial rebuilding [7], there has been some prior work on methods for adapting data structures to the way in which they are being used. Most previous data structure competitive analyses have been directed at simple linked-lists structures, with "move-to-front" heuristics applied [9]. Work on other adaptive data structures includes splay trees [10], which perform a sophisticated move-to-root heuristic, but perform many rotations with each access. There is also the randomized binary search tree of Seidel and Aragon [8], which performs random structural changes with each access and can adapt in an expected, probabilistic sense based on data structure usage.

Energy-Balanced Binary Search Trees. A *dictionary* holds pairs of ordered keys and elements, subject to update and query operations. A common way of implementing the dictionary ADT is to use a binary search tree, which maintains balance by local rotation operations. Typically, such rotations are fast, but if the tree has auxiliary structures, rotations are often slow. Standard binary search trees, such as AVL trees [1], red-black trees [3], scapegoat trees [2], or weight-balanced trees [6], maintain balance, but do not adapt themselves based on the distribution of accesses and updates. Splay trees [10], on the other hand, adapt (in an asymptotic sense), but perform a large number of rotations with each access. Finally, randomized binary search trees [8], have good expected behavior but offer no worst-case guarantees on performance. We describe a simple tree structure that achieves balance without rotations, by using a potential energy parameter stored at each node and partial rebuilding [7]. Our methods are somewhat reminiscent of scapegoat trees [2] and dynamic search trees of Overmars [7]. Our approach does not use any explicit balancing rules. Instead, it uses potential labels on the nodes, which allow it to be adaptive, competitive, and arguably simpler than previous approaches.

An *energy-balanced tree* is a binary search tree T such that each node maintains a parameter n_v, which is the number of elements stored in the subtree rooted at v (including v itself). More importantly, each node v in T also maintains a potential energy parameter, p_v. Insertions and deletions are handled as in standard (unbalanced) binary search trees, with one small modification. Every time we perform an update operation, which traverses a path from the root of T to some node w in T, we increment p_v by 1 for each node v in this path. If there is no node v in this path such that $p_v \geq n_v/2$, then we are done. Otherwise, let v be the highest node in T such that $p_v \geq n_v/2$. We rebuild the subtree rooted at v as a complete binary tree, and we zero out the potential fields of each node in this subtree (including v itself). This is the entire algorithm for performing updates.

THEOREM 1. *The worst-case height of the energy-balanced search tree is $O(\log n)$, and the amortized time for performing an insert or delete in such a tree is also $O(\log n)$.*

Proof. It is enough to show that $n_w < n_v/4$, for any node v with sibling w. So suppose not. Then, since the last rebalance at v and w's parent, z, (when the size of v and w's subtrees were equal) the number of deletions in w's subtree plus the number of insertions in v's subtree must have been at least $3n_v/4$. That is, $p_z \geq 3n_v/4$. At this point in time we have $n_z = n_w + n_v < 5n_v/4$. Hence, $p_z \geq 3n_v/4 > (3/5)n_z$. But this cannot occur, since we would have rebuilt the subtree at z as soon as $p_z > n_z/2$. ∎

Biased Energy-Balanced Search Trees. Our potential energy approach can be further extended to adapt a dictionary to biased distributions of accesses and updates. We augment the tree T in this case so that each node v stores an *access count*, a_v, which counts the number of times that the element stored at v has been accessed. Each time a node is accessed in a search we increment its access count. We also now increment the potential energy parameter of each node on the path from v to the root. We keep the insertion algorithm the same, but now whenever we delete a node v, we increment the potential energy of each node on the path from v to the root by a_v. Let A_v denote the cumulative access counts for all nodes in the subtree rooted at v in T. We do a rebuilding step any time the potential energy of a node rises to be more than a quarter of its access value, i.e., when $p_v \geq A_v/4$. In this adapted binary search tree we rebuild the subtree so that nodes are nearly balanced by their ac-

*Dept. of Computer Science, Johns Hopkins Univ., Baltimore, MD 21218. goodrich.jhu.edu. This work was supported by ARO MURI Grant DAAH04-96-1-0013 and NSF Grant CCR-9732300.

cess counts, that is, we try to balance children by their A_v values. Specifically, there are several top-down approaches (e.g., see [5]), as well as a a simple linear-time bottom-up greedy approach that can guarantee that for any node v with parent z, $A_z \geq 3A_v/2$. For any non-root node v, we use $\hat{A}_v$ to denote the size of the subtree rooted at v plus the weight of the item stored at v's parent z (so $A_z = A_v + \hat{A}_w$, where w denotes v's sibling).

LEMMA 1. *For any node v with sibling w, $\hat{A}_w \geq A_v/8$.*
Proof. Suppose, for the sake of proving a contradiction, that $\hat{A}_w < A_v/8$. Then, since the last rebalance at v and w's parent, z, (when $\hat{A}'_w \geq A'_v/2$, where $\hat{A}'_w$ and A'_v denote the old values of $\hat{A}_w$ and A_v respectively) the total weight of deletions in w's subtree plus the number of insertions and accesses in v's subtree, plus accesses ending at v's parent, must have been at least $3A_v/8$. That is, $p_z \geq 3A_v/8$. At this point in time we have that $A_z = \hat{A}_w + A_v < 9A_v/8$. Hence, we have that $p_z \geq 3A_v/8 > A_z/3$. But this cannot occur, since we would have rebuilt the subtree at z as soon as $p_z > A_z/4$. ■

This lemma immediately implies the following.

THEOREM 2. *An element having current access frequency a is stored at depth $O(\log A/a)$, where A is the current total access frequency of all nodes.*

LEMMA 2. *Let A_i denote the total access counts of all nodes present in the dynamic biased energy-balanced tree (for S) after we perform the i-th operation in S_v. Then $\sum_{i=1}^{m} \log A_i/i$ is $O(m \log \hat{A}/m)$, where $m = |S_v|$ and $\hat{A}$ is the total access counts for all elements referenced in S.*
Proof. Let us assume for the sake of analysis that m is a power of 2, i.e., that $m = 2^k$, for some k. Note that

$$\sum_{i=1}^{m} \log \frac{A_i}{i} \leq \sum_{i=1}^{m} \log \frac{\hat{A}}{i} = \sum_{i=1}^{m} \log \left(\frac{\hat{A}}{m}\right)\left(\frac{m}{i}\right)$$
$$= \sum_{i=1}^{m} \log \frac{\hat{A}}{m} + \sum_{i=1}^{m} \log \frac{m}{i} = m \log \frac{\hat{A}}{m} + \sum_{i=1}^{m} \log \frac{m}{i}.$$

Thus, to establish the lemma we need only bound the the last term above (the summation term). Note that

$$\sum_{i=1}^{m} \log \frac{m}{i} = \sum_{i=1}^{2^k} \log \frac{2^k}{i} \leq \sum_{i=1}^{2^k} \log \frac{2^k}{2^{\lfloor \log i \rfloor}} = \sum_{i=1}^{2^k} k - \lfloor \log i \rfloor$$
$$\leq \sum_{j=1}^{k} j 2^{k-j} = 2^k \sum_{j=1}^{k} \frac{j}{2^j} \leq 2 \cdot 2^k = 2m.$$

■

An oracle, which we call the *biased-tree oracle*, knowing the sequence in advance could construct a static tree based on known access counts, so that the running time for each access or update at a node v is $O(\log \hat{A}/\hat{a}_v)$, where $\hat{a}_v$ denotes the total access count for the element at node v.

THEOREM 3. *The energy-balanced search tree achieves amortized performance for update operations at each node v that is $O(\log \hat{A}/\hat{a}_v)$, which is within a constant factor of the performance achievable by the biased-tree oracle.*
Proof. Let S be a sequence of n dictionary operations and let T be the static tree built by the biased-tree oracle. Consider a subsequence S_v of S formed by all operations that access or update the element at a given node v. Let A_i denote the total access counts of all nodes present in the dynamic adaptable energy-balanced tree (for S) after we perform the i-th operation in S_v. Note that the amortized running time for performing the i-th operation in S_v using the energy-balanced tree is proportional to the future depth of v in the energy-balanced tree, which will be at most $O(\log A_i/i)$. Thus, the amortized time required for our performing all operations in S_v is proportional to at most $\sum_{i=1}^{m} \log A_i/i$, whereas the total time required of the implementation of the biased-tree oracle is proportional to $m \log \hat{A}/\hat{a}_v = m \log \hat{A}/m$, where $m = |S_v|$. By Lemma 2, however, we have that $\sum_{i=1}^{m} \log A_i/i$ is $O(m \log \hat{A}/m)$, which implies that the time performance of the energy-balanced approach on S_v is at most a constant factor more than the time performance achievable by the biased-tree oracle. ■

Thus, biased energy-balanced search trees are efficient, competitive, and simple.

References

[1] G. M. Adel'son-Vel'skii and Y. M. Landis. An algorithm for the organization of information. *Doklady Akademii Nauk SSSR*, 146:263–266, 1962. English translation in *Soviet Math. Dokl.*, **3**, 1259–1262.

[2] I. Galperin and R. L. Rivest. Scapegoat trees. In *Proc. 4th ACM-SIAM SODA*, pages 165–174, 1993.

[3] L. J. Guibas and R. Sedgewick. A dichromatic framework for balanced trees. In *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, Lecture Notes Comput. Sci., pages 8–21. Springer-Verlag, 1978.

[4] M. S. Manasse, L. A. McGeoch, and D. D. Sleator. Competitive algorithms for on-line problems. In *Proc. 20th Annu. ACM Sympos. Theory Comput.*, pages 322–333, May 1988.

[5] K. Mehlhorn. A best possible bound for the weighted path length of binary search trees. *SIAM J. Comput.*, 6(2):235–239, 1977.

[6] J. Nievergelt and E. M. Reingold. Binary search trees of bounded balance. *SIAM J. Comput.*, 2:33–43, 1973.

[7] M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, West Germany, 1983.

[8] R. Seidel and C. R. Aragon. Randomized search trees. *Algorithmica*, 16:464–497, 1996.

[9] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Commun. ACM*, 28:202–208, 1985.

[10] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. ACM*, 32(3):652–686, 1985.