Parallel Subquadratic Work Algorithms for Constructing Approximately Optimal Binary Search Trees

Marek Karpinski *

Lawrence L. Larmore †

Wojciech Rytter ‡

Abstract

A sublinear time sub-quadratic work parallel algorithm for construction of an optimal binary search tree, in a special case of practical interest, namely where the frequencies of items to be stored are not too small, is given. A sublinear time sub-quadratic work parallel algorithm for construction of an approximately optimal binary search tree in the general case is also given. Sub-quadratic work and sublinear time are achieved using a fast parallel algorithm for the *column minima* problem for Monge matrices developed by Atallah and Kosaraju. The algorithms given in this paper take $O(n^{0.6})$ time with n processors in the CREW PRAM model.

A new version of the sequential subquadratic time algorithms for the same problems is also given. New parallel and sequential algorithms for height-limited binary search treees with very small height limitation are presented.

*Dept. of Computer Science, University of Bonn, 53117 Bonn. This research was partially supported by DFG Grant KA 673/4-1, and by ESPRIT BR Grant 7097 and ECUS 030. Email:marek@cs.uni-bonn.de

†Department of Computer Science, University of Nevada, Las Vegas, NV 89154–4019, USA. Partially supported by National Science Foundation grants CCR-9112067 and CCR-9503441. Email:larmore@cs.unlv.edu

‡Institute of Informatics, Warsaw University, 02–097 Warszawa. Partially supported by DFG Grant Bo 56/142-1. Email:rytter@mimuw.edu.pl

1 Introduction

The problem of computing optimal binary search trees (the OBST problem) is especially interesting in a parallel setting, since there is no known $\mathcal{NC}$ algorithm which solves that problem efficiently, and the problem of finding such a parallel algorithm appears to be very hard [4].

There is an $\mathcal{NC}$ algorithm for the special case of alphabetic trees using n^2 processors [12]. The best known $\mathcal{NC}$ algorithms require $O(n^6)$ work for optimal binary search trees and $O(n^2)$ work for approximately optimal binary search trees [4, 15].

Sublinear time parallel algorithms sometimes have much lower total work than $\mathcal{NC}$ algorithms. In [7] a sublinear time algorithm for the OBST problem whose work is very close to quadratic is given. The fastest known sequential algorithm for the OBST problem is the classical algorithm by Knuth [9], which takes quadratic time. The main theorem of [9] uses, without stating it those terms, the Monge property of the matrix of subtree costs. A matrix M has the *Monge property* if, for all $i_0 < i_1$ and $j_0 < j_1$ which are within range, $M_{i_0,j_0} + M_{i_1,j_1} \leq M_{i_0,j_1} + M_{i_1,j_0}$. This is essentially the same as the *quadrangle inequality* introduced by Yao [16] which allowed speedup of certain dynamic programming algorithms.

The problem of developing a sub-quadratic time sequential algorithm for the general OBST problem appears to be very hard. Algorithm for finding approximately optimal binary search trees have been found by Allen, Mehlhorn and Unterauer [2, 13, 14]. The results of this paper are largely based on the algorithm for approximately optimal binary search trees given by Larmore [10].

In this paper we consider the problem in a parallel setting, using the CREW PRAM model of computation. We present sublinear time subquadratic work parallel algorithms for certain special instances of the OBST problem, We shall define an instance to be "special" if the item weights are sufficiently large. We also give sublinear time subquadratic work parallel algorithms which give approximately optimal binary search trees in the general case.

Define a binary search tree to be *ϵ-approximately optimal* if its cost differs by at most ϵ from the cost of the optimal binary search tree. Our main result is:

Theorem 1.1 *There exists an $O(n^{0.6})$-time parallel algorithm using n processors which computes the optimal binary search tree for a special sequence. Furthermore, there exists an $O(n^{0.6})$-time parallel algorithm using n processors which computes an ϵ-approximately optimal binary search tree for a general sequence, where $\epsilon = o(1)$.*

We use terminology from [8], pages 434–435. Let $K_1, \ldots K_n$ be a sequence of n weighted items (keys), which are to be placed in a binary search tree. We are given a sequence α of $2n+1$ weights (probabilities): $q_0, p_1, q_1, p_2, q_2, p_3, \ldots, q_{n-1}, p_n, q_n$ where

- p_i is the probability that K_i is the search argument;
- q_i is the probability that the search argument lies between K_i and K_{i+1}.

Note that $\sum p_i + \sum q_i = 1$. It will be convenient to refer to the *external item* E_i, for $0 \leq i \leq n$, corresponding to the probability q_i.

Given a constant $\delta > 0$, the sequence α is said to be δ-*special* if $p_i + q_i \geq \delta/n$, for each $1 \leq i \leq n$. Let $Tree(\alpha)$ be the set of all full binary weighted trees with n internal nodes, where the i^{th} internal node (in inorder) has weight p_i, and the i^{th} external node (the leaf, in the left-to-right order) has weight q_i. The tree is "full" in the sense that each internal node has exactly two sons. The keys $\{K_i\}$ are to be stored in internal nodes of this binary search tree. The external nodes correspond to intervals between keys. If T is a such a weighted binary search tree then define the *cost* of T to be:

$$cost(T) = \sum \ell(T, v) \cdot weight(v) \tag{1}$$

where the summation is over all nodes of T, and $\ell(T, v)$ is the *level* of the node v in T, defined to be the distance (number of nodes on the path) from the root.

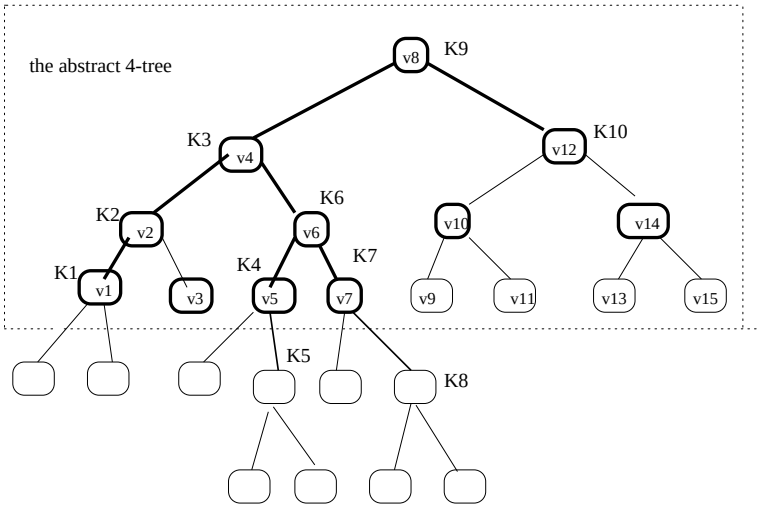

Figure 1: A binary search tree T. Some nodes (boldface) are inside the indicated (dotted rectangle) abstract regular tree $\mathcal{T}_4$. The nodes of $\mathcal{T}_d$ are numbered in in-order. Observe that T does not include all nodes of $\mathcal{T}_d$. The edges of T leading to its internal nodes inside $\mathcal{T}_4$ are in bold. We shall later traverse (in some sense) the subtree consisting of such edges.

Example
Consider the tree T shown in Figure 1. It consists of 10 internal nodes labeled $K_1 \ldots K_{10}$ and 11 external nodes. The contribution of the key K_6 to the cost is $3 \cdot p_6$, for example, since the level of node containing K_6 is 3. Similarly, for example, the contribution of the "unsuccessful search" node E_9 between keys K_8 and K_9 is $6 \cdot q_9$.

Let OPT(α) be the set of trees $Tree(\alpha)$ whose cost is minimal. The OBST problem consists of finding any tree $T \in$ OPT(α). Denote by $obst(i, j)$ the set OPT($q_i, p_{i+1}, q_{i+1}, \ldots, q_{j-1}, p_j, q_j$). The

trees which are elements of this set are said to have *width* $|j-i|$. Let $cost(i,j)$ be the cost of a tree in $obst(i,j)$, and let $weight(i,j) = q_i + p_{i+1} + \ldots + p_j + q_j$, for $i < j$. Let $cost(i,i) = weight(i,i) = q_i$.

The values of $cost(i,j)$ are tabulated in an array. The time to compute all values of *cost* is $O(n^2)$, using Knuth's Theorem [9], essentially making use of the Monge property of *cost*, considered as a matrix. Knuth's algorithm can be easily parallelized by computing all entries on a given diagonal of the array in parallel. The following lemma was essentially shown in [7]. It says that costs of all optimal subtrees of width at most ℓ can be efficiently computed in parallel.

Lemma 1.2 (Parallelization of Knuth's algorithm) *All values $cost(i,j)$ for $|j-i| \leq \ell$ can be computed in $O(\ell \cdot \log(\ell))$ time with $O(n/\log(\ell))$ processors.*

A matrix M has the *Monge property* if, for all $i_0 < i_1$ and $j_0 < j_1$ which are within range, $M_{i_0,j_0} + M_{i_1,j_1} \leq M_{i_0,j_1} + M_{i_1,j_0}$. Monge matrices arise in a large number of applications.

We state several known results concerning Monge matrices.

Lemma 1.3 (Monotonicity of Column Minima) *If $j_0 \leq j_1$, then there exist $i_0 \leq i_1$ such that the minimum of column j_0 of M is at i_0, and the minimum of column j_1 of M is at i_1.*

The following result is by Aggarwal, Klawe, Morey, Shor, and Wilber [1]. The algorithm developed in that paper is whimsically known as the "SMAWK" algorithm, using a permutation of the authors' initials.

Lemma 1.4 *If M is an $n \times m$ Monge matrix, all column minima of M can be found in $O(n+m)$ sequential time.*

In the parallel case, we have the following result by [3].

Lemma 1.5 *If M is an $n \times m$ Monge matrix, all column minima of M can be found in $O(\log n \log m)$ time by $n/\log n$ processors, using the CREW PRAM model of computation.*

2 A general structure of the exact algorithm

The main phase of our algorithm uses a form of dynamic programming quite different from the usual ones for optimal binary search trees (bottom-up computation of the costs of optimal subtrees). The new concept of a "partial tree" is introduced. The costs of all partial trees are computed by processing the potential nodes of the trees in in-order. These potential nodes are contained in a tree which we call an "abstract tree." It is possible that not all of the nodes of the abstract tree correspond to nodes in the optimal binary search tree.

Let $d > 0$ be a given integer. The *abstract d-tree* $\mathcal{T}_d$ is a full regular binary tree which consists of all possible nodes at level at most d, and no nodes at higher levels. See Figure 1. Note that $\mathcal{T}_d$ has $m = 2^d - 1$ nodes, which we label $v_1, \ldots v_m$ in in-order. For example, the nodes of $\mathcal{T}_4$ as shown in Figure 1, are labeled $v_1 \ldots v_{15}$. We shall identify the nodes at levels at most d of any binary tree

with nodes of $\mathcal{T}_d$. For example, in Figure 1, all internal nodes of T except K_5 and K_8 are identified with nodes of $\mathcal{T}_d$, as are 3 of its 11 external nodes. Some of the nodes of $\mathcal{T}_d$, namely v_9, v_{11}, v_{13}, and v_{15} are not identified with nodes in T.

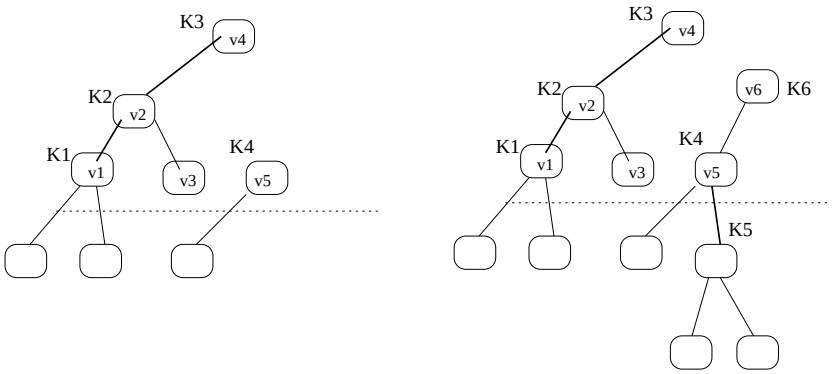

Figure 2: Two partial trees: $T' = Partial_T(v_4, K_4)$ and $T'' = Partial_T(v_6, K_6)$. $cost(T') = 4q_0 + 3p_1 + 4q_1 + 2p_2 + 3q_2 + p_3 + 4q_3 + 3p_4$. The difference of costs between these trees is $(level(v_5) + 1) \cdot cost(4, 5) + level(v_6) \cdot p_6$.

Partial subtrees. Assume T is a binary search tree and $v \in \mathcal{T}_d$ is identified with an internal node of T containing the key K_i. Then $T' = Partial_T(v, K_i)$ is a subtree of T which consists of all vertices(internal and external) of T preceding the node v in in-order, together with v. We say that T' is a *partial tree terminating in* (v, K_i). Two partial trees are illustrated in Figure 2.

The cost of the partial tree T' (written $partial\_cost(T')$) is the sum of the path weights for all nodes in T'. Define:

$$partial\_cost(v, i) = \min\{partial\_cost(T') \ : T' \text{ is a partial tree terminating in } (v, K_i)\}. \qquad (2)$$

Our main algorithm depends on two parameters, ℓ and d, and consists of three phases.

ALGORITHM MAIN:

Preprocessing-Phase: parallel implementation of Knuth's algorithm.

Compute costs of optimal subtrees of width at most ℓ.
Comment: can be done in parallel time $O(\ell)$ due to Lemma 1.2

Basic-Phase: computation of optimal costs of partial subtrees.

Assume the nodes of the tree $\mathcal{T}_d$ are listed in in-order $v_1, \ldots, v_m$.
for $k = 1$ **to** m **do**
for each $i = 1 \ldots n$ **do in parallel**
compute $partial\_cost(v_k, i)$ using the parallel algorithm
of [3] for the corresponding column minima problem.

Construction-Phase: construction of an optimal binary search tree.

$global\_cost := \min\{partial\_cost(v_k, n) + (level(v) + 1) \cdot q_n \ : \ v \in \mathcal{T}_k \}$;
Comment: $global\_cost$ is the cost of an optimal tree;
$w :=$ a node $v \in \mathcal{T}_k$ for which minimum is achieved;
construct an optimal tree knowing w and the table $partial\_cost$.

3 Analysis of the algorithm MAIN

The essential part of the algorithm is *Basic-Phase*. We derive reccurence equations, as in dynamic programming, to compute the table *partial_cost*. First we introduce the relation "$\Rightarrow$" between the nodes of the abstract tree $\mathcal{T}_d$. The relation $u \Rightarrow v$ means that there is some binary tree T for which a node identified with v is the immediate in-order successor, in T, of a node identified with u. More formally: let u_1, w_1 be, respectively, the left and right sons of u in $\mathcal{T}_d$. Let $u_2, u_3, \ldots$ be the rightmost branch starting at u_1 and let $w_2, w_3, \ldots$ be the leftmost branch starting at u_1. Then $u_i \Rightarrow v$ and $v \Rightarrow w_i$ for each i (see Figure 3).

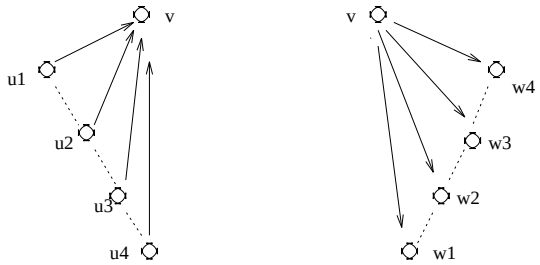

Figure 3: The arrows show the relation $\Rightarrow$.

If $u \Rightarrow v$, we say u is a *predecessor* of v, and v is a *successor* of u. The cost of an optimal partial tree terminating in a given node v depends on the cost of a partial tree terminating in a predecessor

of v. Let $\ell(u,v) = \max\{level(u), level(v)\} + 1$. We introduce two auxiliary tables *partial_cost_1* and M_v defined by recurrence equations as follows:

$$partial\_cost\_1(v,i) = \min\{partial\_cost(u,i-1) + level(v) \cdot p_i + \ell(u,v) \cdot q_{i-1} \ : \ u \Rightarrow v\} \qquad (3)$$

Assume $u \Rightarrow v$ and u or v is at the bottom level, *i.e.*, $\ell(u,v) = d+1$. Then for each $1 \leq i < j$ define:

$$M_v(i,j) = partial\_cost(u,i) + level(v) \cdot p_j + cost(i-1,j) + weight(i-1,j) \cdot d \qquad (4)$$

If $i \geq j$ then define $M_v(i,j) = \infty$. Let $ColMin(M_v, i)$ be the smallest value in the i^{th} column of M_v. The *basic dynamic programming recurrence* for computing *partial_cost* is as follows:

$$partial\_cost(v,i) = \min\{partial\_cost\_1(v,i),\ ColMin(M_v, i)\} \qquad (5)$$

Example Consider the partial trees T' and T'' in Figure 2. Assume these partial trees are optimal. In this case $u = v_5$ and $v = v_6$. The difference between costs of these trees is $\ell(v_5, v_6) \cdot cost(4,5) + level(v_6) \cdot p_6$. In other words, this difference is $M_{v_6}(4,6)$. The column minimum is realized in the 4^{th} row of the 6^{th} column. We also have $partial\_cost(v_6, 6) = partial\_cost(v_5, 4) + ColMin(M_{v_6}, 6)$.

Lemma 3.1 (key lemma)
1. The matrix M_v satisfies the Monge condition.
2. For a given v, the values $ColMin(M_v, i)$, for all $1 \leq i \leq n$, can be computed in $O(\log^2 n)$ time with $n/\log n$ processors.

Proof. (1) By Lemma 2.1 of [10], the matrix $\{cost(i-1,j)\}$ has the Monge property. It is simple to verify that $\{weight(i-1,j)\}$ is also Monge. The other two terms are trivially Monge since they depend on only one component. Finally, the sum of Monge matrices is Monge.

(2) All column minima of an $n \times n$ Monge matrix can be computed in $O(\log^2 n)$ time with $n/\log n$ processors by a simple divide-and-conquer algorithm. ∎

Lemma 3.2 (Complexity of MAIN) *Assume that the total weight of each segment of ℓ consecutive items is at least Δ. Then an optimal binary search tree can be computed in $O(\max\{\ell \cdot \log n, 2^{\log_\phi(\frac{1}{\Delta})}\})$ parallel time with n processors.*

Proof.
The partial costs can be computed by traversing the tree $\mathcal{T}_d$, for $d = \lceil \log_\phi(\frac{1}{\Delta}) \rceil + 1$, in in-order and applying the basic dynamic programming recurrence. The main point is that the values *ColMin* can be computed for a given node in $O(\log n)$ time with n processors, by Lemma 3.1. This proves the following claim:

Claim 1. Assume the costs of all optimal subtrees rooted below level d are computed. Then an optimal binary search tree can be constructed in $O(2^d \log n)$ time with n processors.

We next use a combinatorial fact shown in [5] and [6] and expressed by the following claim:

Claim 2. Let ϕ be the *golden ratio* ($\phi = \frac{\sqrt{5}+1}{2} \approx 1.62$) and let $T \in OPT(q_0, p_1, q_1, \ldots, p_n, q_n)$. If v is an internal node of T and the weight of all items contained in the subtree rooted at v is Δ, then $level_T(v) < \log_\phi(\frac{1}{\Delta}) + 2$.

Proof. (of Claim 2). Let $F_1 = 1$, $F_2 = 1$, $F_3 = 2$, *etc.*, be the Fibonacci numbers. By [6], a subtree whose root is at level h can have weight at most $2/F_{h+2}$. Since $F_n > 2\phi^{n-4}$ (see [8] exercise 4, pg 18) we are done. ∎

The structure of the computation is shown in Figure 4. First the optimal costs of subtrees of width ℓ are computed. Then (in *Basic-Phase*) the partial costs are computed in $O(2^d \log n)$ time with n processors (see Claim 1). Efficiency is gained by applying the parallel algorithm for the column minima problem.

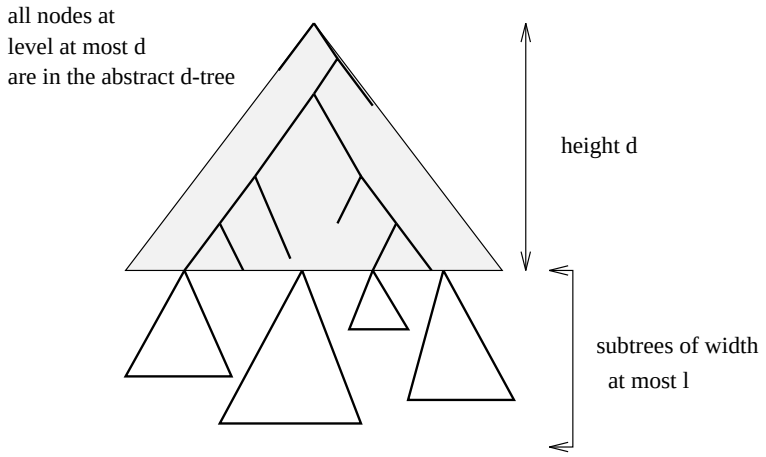

Figure 4: The structure of a binary search tree: $d = \lceil \log_\phi(\frac{1}{\Delta}) \rceil + 2$, where Δ is the smallest total weight of a segment of ℓ consecutive items. The abstract d-tree $\mathcal{T}_d$ is shaded.

The first phase runs in $O(\ell \cdot \log n)$ time with $O(n\ell)$ work, and the second phase runs in $O(2^d \log n)$ time with $O(2^d n)$ work.

Finally, the binary search tree is reconstructed using pointers that are saved during computation of *cost* and M. ∎

4 The proof of the main results

In this section we prove our main results, as two separate theorems. The proofs consist of manipulating the parameters d, ℓ, and Δ. Let $\beta = 1/(1 + \log_2 \phi)$. We have $\beta \approx 0.59023$. Hence $n^\beta \log n = O(n^{0.6})$. We restate Theorem 1.1 precisely:

Theorem 4.1 *Assume $q_{i-1} + p_i + q_i \geq \frac{\delta}{n}$ for each i, where $\delta > 0$ is a constant. Then an optimal binary search tree can be computed in $O(n^{0.6})$ parallel time with $O(n)$ processors.*

Proof. We apply Lemma 3.2 with $\Delta = \frac{\delta \ell}{n}$. Let $d = \log_\phi(\frac{n}{\ell})$. The work in the first phase is $O(n\ell)$ and in the second phase it is $O(2^d n)$. The algorithm has the smallest minimal work if these subworks are nearly equal. This occurs when $\log_\phi(\frac{n}{\ell}) = \log_2(\ell)$. It can be calculated that in this case $d = \log_2(n) \cdot \beta$. Thus $2^d \log n = O(n^{0.6})$. ∎

Theorem 4.2
1. Let $a > 1$. There is a parallel $O(n^{1+a\cdot\beta} \log n)$-time n-processor algorithm which constructs an $(n^{1-a} \log n)$-approximately optimal binary search tree.
2. There is a parallel $O(n^{0.6})$-time n-processor algorithm which constructs an $o(1)$-approximately optimal binary search tree.

Proof. Let $\delta = n^{-a}$.

Claim 1
If each item has weight at least δ we can compute the optimal binary search tree in parallel $O(n^{1+a\cdot\beta} \log n)$ time with n processors.
Proof. (of the claim) Take $\ell = n^{a\cdot\beta}$, $\Delta = \ell \cdot \delta$ and apply Lemma 3.2. ∎

We remove a pair consisting of K_i and E_i provided $q_{i-1} + p_i + q_i < \delta$. Iterate this process until $q'_{i-1} + p'_i + q'_i \geq \delta$ for all i in the remaining sequence. Construct an optimal binary search tree T' for the remaining items using the algorithm from Claim 1. The tree T' has height $O(\log n)$ since the weight of each item is sufficiently large. The cost of T' does not exceed the cost of an optimal tree for the whole sequence.

We now attach the deleted items to T', as follows. Suppose $K_i, E_i \ldots K_j, E_j$ is a maximal list of consecutive deleted items. Replace E_{i-1} in T' by an almost regular binary search tree whose items are $E_{i-1}, K_i, \ldots K_j, E_j$. See Figure 4.

We increase the cost by $O(n \cdot n^{-a} \cdot \log n)$, which is $O(n^{1-a} \log n)$. This proves point 1.

We can take a very close (from below) to $0.6/\beta$ and achieve time $O(n^{0.6})$ with n processors and $o(1)$-approximation. ∎

5 Short height limited trees

The fastest known sequential algorithm for optimal height-limited binary search trees requires $O(n^2 L)$ time, where L is the height. Below, we give a new algorithm which improves this result for small values of L.

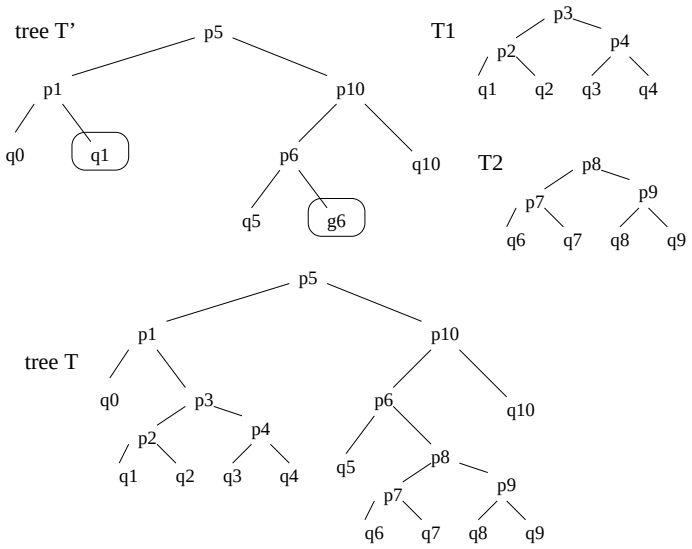

Figure 5: Assume that we have two runs of small items $K_2, \ldots, K_4$ and $K_7..K_9$. IF T' is an optimal tree for the remaining items then regular binary trees T1 and T2 (for the runs of small items) are attached at the circled nodes of T' to form the final approximately optimal tree T.

Theorem 5.1 *If $L \leq 2\log_2 n - \gamma$, where $\gamma > 0$ is a constant, then there is a sequential algorithm which computes the optimal binary search tree of height L in subquadratic time.*

We sketch the proof. The first step is to compute all optimal trees of height at most $L/2$ for all pairs where $|j-i| < 2^{L/2}$. It is important to use the fact that $\left\{cost^h(i,j)\right\}$ is a Monge matrix for fixed h, where $cost^h(i,j)$ is cost of an optimal binary search tree of the sequence $q_i, p_{i+1}, \ldots p_j, q_j$ subject to the restriction that its height is at most h. The second step is a sequential time implementation of the construction phase of the main algorithm from Section 2.

There is a corresponding parallel algorithm.

Theorem 5.2 *If $L \leq 2\log_2 n - \gamma$, where $\gamma > 0$ is a constant, then there is a parallel algorithm which computes the optimal binary search tree of height L in sublinear time with subquadratic work.*

References

[1] A. Aggarwal, M. Klawe, S. Moran, P. Shor, and R. Wilber, Geometric applications of a matrix-searching algorithm, *Algorithmica* **2** (1987), pp. 195–208.

[2] B. Allen, Optimal and near-optimal binary search trees, *Acta Inform.* **18** (1982), pp. 255–263.

[3] M. J. Atallah, S. R. Kosaraju, Parallel computation of row minima for monotone matrices, *Journal of Algorithms* **13** (1992), pp. 394–413.

[4] M. J. Atallah, S. R. Kosaraju, L. L. Larmore, G. L. Miller, and S-H. Teng, Constructing trees in parallel, *Proc. 1^{st} ACM Symposium on Parallel Algorithms and Architectures* (1989), pp. 499–533.

[5] R. Güttler, K. Mehlhorn, and W. Schneider, Binary search trees: average and worst case behavior, *Elektr. Informationsverarb Kybernetik* **16** (1980), pp. 579–591.

[6] D. S. Hirschberg, L. L. Larmore, and M. Molodowitch, Subtree weight ratios for optimal binary search trees, TR 86-02, ICS Department, University of California, Irvine (1986).

[7] M. Karpinski and W. Rytter, On a sublinear time parallel construction of optimal binary search trees, *Proceedings of the 19^{t}h International Symposium on Mathematical Foundations of Computer Science*, LNCS 841 (ed. I. Privara, B. Rovan, P. Ruzicka) (1994), pp. 453–461.

[8] D. E. Knuth, *The Art of Computer Programming*, Addison–Wesley (1973).

[9] D. E. Knuth, Optimum binary search trees, *Acta Informatica* **1** (1971), pp. 14–25.

[10] L. L. Larmore, A sub-quadratic algorithm for constructing approximately optimal binary search trees, *Journal of Algorithms* **8** (1987), pp. 579–591.

[11] L. L. Larmore and W. Rytter, Efficient sublinear time parallel algorithms for dynamic programming problems and context-free recognition. *STACS'92, 9th Symposium on Theoretical Aspects of Computer Science, Lect. Notes in Comp. Science*, Edited by A. Finkel and M. Jantzen, Springer Verlag (1992), pp. 121–132.

[12] L. L. Larmore, T. M. Przytycka, and W. Rytter, Parallel construction of optimal alphabetic trees, *Proceedings of the 5th ACM Symposium on Parallel Algorithms and Architectures* (1993), pp. 214–223.

[13] K. Mehlhorn, Nearly optimal binary search trees, *Acta Informatica* **5** (1975), pp. 287–295.

[14] K. Unterauer, Dynamic weighted binary search trees, *Acta Informatica* **11** (1979), pp. 341–362.

[15] W. Rytter, Efficient parallel computations for some dynamic programming problems, *Theoretical Comp. Sci.* **59** (1988), pp. 297–307.

[16] F. F. Yao, Efficient dynamic programming using quadrangle inequalities, *Proceedings of the 12th ACM Symposium on Theory of Computing* (1980), pp. 429–435.