Math. Systems Theory 15, 1–16 (1981)

Mathematical Systems Theory

Lower Bounds on the Efficiency of Transforming Static Data Structures into Dynamic Structures

Kurt Mehlhorn

Fachbereich 10, Angewandte Mathematik und Informatik, Universität des Saarlandes, 6600 Saarbrücken, West Germany

Abstract. In this paper we study the efficiency of general methods for converting static data structures to dynamic structures. The efficiency is measured in terms of two quantities: query time and update time penalty factors. We provide lower bounds on the trade-off between these quantities and show certain known transforms to be essentially optimal.

Introduction

Recently several efforts were made to develop general methods for turning static solutions (only searches are supported) to problems into dynamic ones (insertions and [maybe deletions] are also supported). Bentley [1] and later Saxe and Bentley [4] made the important observation that a general approach to "dynamization" is especially relevant to the class of so-called decomposable searching problems. A searching problem can be viewed as a problem in which one asks a question about an arbitrary object x of type T_1 and a (static or dynamic) set of objects (from now on called points) of type T_2, with an answer of type T_3. We can denote such a query as:

$$Q: T_1 \times 2^{T_2} \to T_3$$

For instance, in a Member query, T_1 and T_2 are identical and T_3 is boolean.

Definition 1.1. A searching problem $Q: T_1 \times 2^{T_2} \to T_3$ is called decomposable if there exists a "trivially" computable operator $\square$ on the elements of T_3, satisfying:

$$\forall A, B \in 2^{T_2} \quad \forall x \in T_1 \qquad Q(x, A \cup B) = \square(Q(x, A), Q(x, B))$$

0025/5661/81/0015-0001$03.20

For instance, a Member query is decomposable ($\square = or$). Saxe/Bentley [4] and later v. Leeuwen/Wood [2], Mehlhorn/Overmars [2], Overmars/v. Leeuwen [3], Willard [6] presented several general methods for dynamizing static data structures. The basic principle is the following:

Suppose we have a static data structure for some searching problem which supports queries on point sets of n elements in time $Q_S(n)$ and which can be constructed for a point set of n elements in time $P_S(n)$. We will assume (as is customary in the field) that functions $Q_S(n)$ and $P_S(n)/n$ are nondecreasing.

Transformations from static to dynamic data structures adhere to the following principles (cf. [4], p. 324–328, for a very detailed discussion):

a) A set S of n elements is represented by static structures $T_1, \ldots, T_r$ for sets $S_1, \ldots, S_r$ where

$$S = \bigcup_{i=1}^{r} S_i.$$

b) A search (query) about set S is answered by searching sets $S_1, \ldots, S_r$ and then putting the individual answers together to a final answer.

c) A new element x is inserted by creating a static structure T for singleton set $\{x\}$ and then (maybe) taking some existing structures $T_{i_1}, T_{i_2}, \ldots, T_{i_s}$ for sets $S_{i_1}, \ldots, S_{i_s}$ and building new structures for a repartioning of those sets.

Remark. Note that we do not require the sets $S_1, \ldots, S_r$ in a) to be pairwise disjoint, as Bentley and Saxe did in their original definition of decomposable searching problems. Of course, this greater flexibility in set representation will restrict the class of decomposable searching problems, e.g. the membership problem is decomposable in this restricted sense, however, the problem count (number of appearances in a multi-set) is not. On the other hand, this greater flexibility might lead to more efficient transforms for the restricted class of problems. One of the outcomes of this paper is that this is not the case.

Different transformations are obtained by different strategies for dividing a large set into pieces.

The performance of a transformation is measured by two quantities: the query penalty factor $k(n)$ and the update penalty factor $h(n)$.

Let r_i be the number of blocks in the representation of set S after the i-th insertion, and let m_i be the total size of the sets handled after the i-th insertion (for a precise definition cf. section II). Then

$$k(n) = \max\{r_i;\ 1 \leq i \leq n\}$$

$$h(n) = \sum_{i=1}^{n} m_i/n.$$

The names for quantities $k(n)$ and $h(n)$ are justified by the observation that $k(n) \cdot Q_S(n)$ bounds the time for a query when at most n points are in the structure and $h(n) \cdot P_S(n)$ bounds the cost for inserting the first n points. (Note

that this cost is

$$\sum_{i=1}^{n} P_S(m_i) = \sum_i m_i P_S(m_i)/m_i \leq \sum_i m_i P_S(n)/n = h(n) \cdot P_S(n).)$$

The pair $(k(n), h(n))$ is called the characteristic of the transformation.

Transformations with various characteristics were presented in Saxe/Bentley [4] and Overmars/v. Leeuwen [3], e.g. the binary transform with characteristic $(\log n, \log n)$ the k-binomial transform with characteristic $((k!n)^{1/k}, k)$....

Saxe/Bentley [4] also began a study of lower bounds on the efficiency of transformations, i.e. a study of the trade-off between update and query penalty factor. More specifically, for a restricted class of transformations (called arboreal strategies; a definition can be found in section II below) they prove lower bounds on update penalty factor as a function of query penalty factor. They also pose the problem to generalize their lower bounds to a general class of transformations. We answer this challenge in this paper and show:

Theorem. *There is a constant $c>0$ such that for any transformation with characteristic $(k(n), h(n))$:*

$$h(n) \geq c \cdot \hat{h}(n) \qquad \textit{for all } n$$

where

$$\hat{h}(n) = \begin{cases} \log n / \log[k(n)/\log n] & \textit{if } k(n) \geq 2 \log n \\ k(n) n^{1/k(n)} & \textit{if } k(n) \leq 2 \log n \end{cases}$$

This theorem implies that certain known transformations such as the binary transform and the k-binomial transform are optimal up to a constant factor. More generally, this theorem has to be seen in context with the result of Mehlhorn/Overmars [2], in which the corresponding upper bound is proven.

Theorem (Mehlhorn/Overmars [2]). *Let $k(n)$ be a function such that $k(n)/\log n$ is either increasing or decreasing. Then there is a transform with characteristic $(O(k(n)), O(\hat{h}(n)))$, where $\hat{h}$ is defined as above. Moreover, this transform can be obtained in a systematic way.*

Together, the theorems characterize the trade-off between update- and query penalty factor up to a constant factor. More specifically, Table 1 describes some sample applications of the main theorem. One remark is in order about the abundance of "big-omegas" in Table 1. The proof of the main theorem shows that c can be taken to be $1/512$. However, working through the proof for any particular function $f(n)$ gives much smaller values of c (cf. proof of corollary 3). Also for the sample applications transformations with characteristic $(k(n), O(\hat{h}(n))$ are known.

Table 1. Sample applications of main theorem

Query Penalty	*Update Penalty*
1	$\Omega(n)$
k	$\Omega(kn^{1/k})$
$\log\log n$	$\Omega[(\log\log n)\cdot n^{1/\log\log n}]$
$\log n$	$\Omega(\log n)$
$n^{1/k}$	$\Omega(k)$
n	$\Omega(1)$

We should also mention that the Mehlhorn/Overmars construction yields arboreal transformations; thus the two theorems above imply that for every transformation with characteristic $(k(n), h(n))$ there is an arboreal transformation with characteristic $(O(k(n)), O(h(n)))$ (provided that $k(n)$ is sufficiently smooth). This shows that arboreal transformations is all one needs to consider.

Finally, we should mention that our lower bounds for general transformations are slightly weaker (by a constant factor of 4) than the results of Bentley and Saxe for arboreal transformations. We believe that this loss of a factor of 4 could be avoided by a more careful analysis.

Efficiency of Transforms

In section I we outlined the general principles underlying all (existing) transformations from static to dynamic structures. We will repeat the principles in more detail here in order to derive from them a model of computation.

a) a set $S=\{x_1,\ldots,x_n\}$ of n elements is represented by (static structures for) sets $S_1,\ldots,S_r$ where

$$S=\bigcup_{i=1}^{r} S_i.$$

b) a search (query) about set S is answered by searching sets $S_1,\ldots,S_r$ and then putting the individual answers together to a final answer.

c) a new element x_{n+1} is inserted by constructing (a static structure for) singleton set $\{x_{n+1}\}$.

d) at some points the transformation may want to reshape the current representation of the set. It has two means for doing so:

d1) It can destroy one of the existing structures, if the remaining sets suffice to represent set S.

d2) It can take some of the existing sets, say $T_1,\ldots,T_k$, and repartition them; more precisely, it can construct new sets $S_1,\ldots,S_h$, where $S_1\cup\cdots\cup S_h\subseteq T_1\cup\cdots\cup T_k$. Also it may want to delete some of the sets $T_1,\ldots,T_k$, say $T_{i_1},\ldots,T_{i_l}$, after adding the new sets.

With the identification $i\Leftrightarrow x_i$, a transformation handles sets of integers using the following instruction set. We use $S, S_1, S_2,\ldots, T, T_1, T_2,\ldots$ to denote sets of integers.

I) (insertion): construct $\{i\}$, with meaning: if the class $\{S_j;\ j\in I\}$ of sets exists before execution then the class $\{S_j;\ j\in I\}\cup\{\{i\}\}$ exists afterwards.

II) (reshaping): construct $S'_1,\ldots,S'_h$ from $T_1,\ldots,T_k$ delete $T_{i_1},\ldots,T_{i_l}$, with meaning: if the class $\{S_j;\ j\in I\}$ exists before execution and $T_1,\ldots,T_k\in\{S_j;\ j\in I\}$ and $S'_1\cup\cdots\cup S'_h\subseteq T_1\cup\cdots\cup T_k$ then the class $(\{S_j;\ j\in I\}\cup\{S'_1,\ldots,S'_h\})-\{T_{i_1},\ldots,T_{i_l}\}$ exists after execution.

III) (reshaping): delete T with meaning: if the class $\{S_j;\ j\in I\}$ exists before execution, then the class $\{S_j;\ j\in I\}-\{T\}$ exists afterwards.

A transformation is an infinite straightline program over the above instruction set satisfying the additional restriction that the integers $1,2,3,\ldots$ are inserted in order. This leads to the following definition.

Definition. Let n be an integer. A program $P=I_1,\ldots,I_m$ of m instructions of the form above is admissible for n if a) for every $i, 1\leq i\leq n$, there is exactly one instruction Construct $\{i\}$

b) if $I_s=$Construct $\{i\}$ and $I_t=$Construct $\{j\}$ and $i<j$ then $s<t$, i.e. elements are inserted in order

c) if the class $\{S_i;\ i\in I\}$ of sets exists after execution of P then $\cup S_i=\{1,\ldots,n\}$.

Remark. If P is admissible for n then any prefix of P is admissible for some $i\leq n$.

Definition. a) An infinite program $P=I_1, I_2,\ldots$ is admissible if for every n there is a prefix P_n of P which is admissible for n.

b) a *transformation* is an infinite admissible program.

Example. The following program is 3 admissible; sets $\{2,3\}$, $\{1,2\}$ exist after execution

Construct $\{1\}$;
Construct $\{2\}$;
Construct $\{3\}$;
Construct $\{1,2\}$ from $\{1\}$, (2), delete $\{1\}$;
Construct $\{2,3\}$ from $\{2\}$, $\{3\}$, delete $\{2\}$; delete $\{3\}$

Bentley and Saxe ([4]) considered a subclass of transformations, called arboreal transformations.

Definition. A transformation $P=I_1, I_2,\ldots$ is an *arboreal transformation*, if

a) there are no instructions of type III and each instruction of type II is of the form

construct $T_1\cup\cdots\cup T_k$ from $T_1,\ldots,T_k$, delete $T_1,\ldots,T_k$.

In particular, this implies that the sets which exist at any one time during the execution of the program are pairwise disjoint,

b) each set constructed is a contiguous subset (interval) of $\mathbb{N}$

c) unions are executed as early as possible, i.e. in an instruction

construct $T_1\cup\cdots\cup T_k$ from $T_1,\ldots,T_k$, delete $T_1,\ldots,T_k$

at least one T_i is a singleton set, say $\{j\}$, and that singleton set was built by the immediately preceding instruction.

Next we define the cost and the width of a program P.

Definition. Let $P=I_1,\ldots,I_m$ be a finite program.

a) The cost $c(P)$ is defined as

$$c(P)=\sum_{i=1}^{m} c(I_i)$$

where $c(I)$ is zero for an instruction of type I or III, and

$$c(I)=\sum_{j=1}^{h}|S_j|+\sum_{j=1}^{k}|T_j|$$

for an instruction of type II.

b) The width w_t of P after instruction I_t of P is the number of sets which exist after execution of instruction I_t. The width $w(P)$ is defined as

$$w(P)=\max\{w_t;\ 1\leq t\leq m\}.$$

The width of program P is the maximal number of sets which exist at any point during the execution of P. $w(P)$ corresponds to the query time penalty factor. The cost of an instruction is defined as the cost of reading sets T_i and constructing sets S_j, $1\leq i\leq k$, $1\leq j\leq h$. Thus $c(P)/n$ corresponds to the update time penalty factor, where P is admissible for n.

At this point we are able to state our main theorem: A tradeoff between cost and width of admissible programs.

Theorem 1. *There is a function $L_k(n)$ (of two arguments k and n) such that for all k, n*

1) *If P is admissible for n and $w(P)=k$ then $c(P)\geq\frac{2}{4}\cdot L_k(n)$*

2) *if $n=\binom{h+1+k}{k}$ for some h then*

$$L_k(n)=\frac{h\cdot k}{h+k+1}\binom{h+k+2}{k+1}=\frac{k}{k+1}\,\frac{h+k+2}{h+k+1}\cdot h\cdot n$$

3) *$L_k(n)/n$ is non-decreasing in n.*

A proof of theorem 1 can be found in section III. Corollaries 1 to 3 below describe the content of theorem 1 more explicitely. Theorem 1 extends a similar result by Bentley and Saxe. They showed that every *arboreal* program P which is admissible for n and has $w(P)=k$ must have $c(P)\geq\tilde{L}_k(n)$, where

$$\tilde{L}_k(n)=\frac{k}{k+1}\cdot(h+1)\cdot n$$

for $n=\binom{h+1+k}{k}$. Note that $L_k(n)$ above and their $\tilde{L}_k(n)$ is essentially the same. The factor $2/4$ in our bounds comes from two places. Firstly, we defined cost differently; the cost of instruction construct $T_1 \cup \cdots \cup T_k$ from $T_1,\ldots,T_k$ delete $T_1,\ldots,T_k$ is $2\cdot\Sigma|T_i|$ in our case and only $\Sigma|T_i|$ in their case. This accounts for the factor of 2. Secondly, in our lower bound proof we loose a factor of 4 in the proof of lemma 1. We believe that more careful proofs of lemmas 1 to 3 will avoid that loss.

Corollary 1. *There is a constant $c>0$ ($c=1/512$ will do) such that for every transformation P, P_n the shortest prefix of P including the instruction construct $\{n\}$, $k(n)=w(P_n)$ the width (query penalty factor) of P_n, $h(n)=c(P_n)/n$ the normalised cost (update penalty factor):*

$$h(n) \geq c\cdot\hat{h}(n)$$

where

$$\hat{h}(n)=\begin{cases}\log n/\log[k(n)/\log n] & \text{if } k(n)>2\log n\\ k(n)n^{1/k(n)} & \text{if } k(n)\leq 2\log n\end{cases}$$

Proof. Consider any n. Let P_n be the shortest prefix of P including the statement construct $\{n\}$. Then P_n is admissible for n. Let $k=k(n)=w(P_n)$ be the width of program P_n and let $n\cdot h(n)=c(P_n)$ be the cost of program P_n. Let h be such that $n'\equiv\binom{h+1+k}{k}\leq n<\binom{h+2+k}{k}$.

$$\begin{aligned}\text{Then } h(n) &= c(P_n)/n\\ &\geq 2\cdot L_k(n)/(4n) && \text{part 1) of theorem 1}\\ &\geq 2\cdot L_k(n')/(4n') && \text{part 3) of theorem 1}\\ &\geq 2\cdot(k/(k+1))\cdot((h+k+2)/(h+k+1))\cdot h\cdot n'/(4n') && \text{by part 2) of theorem 1}\\ &\geq h/4 && \text{since } k\geq 1\end{aligned}$$

It remains to derive lower bounds for h in terms of n and $k=k(n)$. We distinguish three cases according to the relative size of k and $\log n$. In the following derivations we will make frequent use of the inequality $m!\geq(m/e)^m$, $e=2.71\ldots$, which follows immediately from Sterling's formula.

Case 1. $k=k(n)<(\log n)/3$. We will show $h\geq\hat{h}(n)/(2e)$ *for* $n\geq 64$.
Assume first, that $h\leq k+2$. Then

$$n<\binom{h+2+k}{k}<2^{2(k+2)}\leq n \qquad \text{for } n\geq 64,$$

a contradiction. Hence $h \geq k+2$ for $n \geq 64$. Assume next, that $h \leq \hat{h}(n)/(2e)$. Then

$$n < \binom{h+2+k}{k} \leq \binom{2h}{k} \leq \frac{(2h)^k}{k!} \leq \left(\frac{2eh}{k}\right)^k$$

$$\leq \left(\frac{\hat{h}(n)}{k}\right)^k = n, \text{ a contradiction. This shows } h \geq \hat{h}(n)/(2e) \text{ for } n \geq 64 \text{ in case 1.}$$

Case 2. $(\log n)/3 \leq k = k(n) \leq 2 \log n$. Then $\hat{h}(n) \leq 16 \log n$. It therefore suffices to show $h \geq (\log n)/8 - 2$. Assume otherwise. Then $h \leq (\log n)/8 - 2$ and hence

$$n < \binom{h+2+k}{k} \leq \binom{(\log n)/8 + k}{k}$$

$$\leq \binom{(\log n)/8 + 2\log n}{2\log n} \qquad \text{since } k \leq 2\log n$$

$$\leq \binom{(2+1/8)\log n}{(1/8)\cdot \log n} \leq \frac{[(2+1/8)\log n]^{(\log n)/8}}{[(\log n)/8]!}$$

$$\leq \left[\frac{e(2+1/8)\log n}{(\log n)/8}\right]^{(\log n)/8} \leq n^{(\log(17e))/8} \leq n,$$

a contradiction. This finishes the proof in case 2.

Case 3. $k = k(n) > 2 \log n$. We will show $h \geq \hat{h}(n)/6 - 2$. Assume otherwise. Then $h + 2 \leq \hat{h}(n)/6$ and hence

$$n < \binom{h+2+k}{k} \leq \binom{2k}{h+2} \qquad \text{since } h+2 \leq \hat{h}(n)/6 \leq (\log n)/6 \leq k$$

$$\leq \frac{(2k)^{h+2}}{(h+2)!} \leq \left(\frac{2ek}{h+2}\right)^{h+2} \leq \left(6ek(n)/\hat{h}(n)\right)^{\hat{h}(n)/6}$$

Let $s = k(n)/\log n$. Then $\hat{h}(n) = \log n / \log s$ and hence

$$6ek(n)/\hat{h}(n) \leq 6e(s \cdot \log n)/[\log n/\log s]$$

$$\leq 17 s \cdot \log s$$

$$\leq s^6 \qquad \text{since } s \geq 2$$

Thus $n < (6ek(n)/\hat{h}(n))^{\hat{h}(n)/6}$

$$\leq s^{6 \log n/(6 \log s)} = n, \text{ a contradiction.}$$

This finishes the proof in case 3.

In either case we have shown $h \geq c' \cdot \hat{h}(n) - 2$ for some constant $c' > 0$. This proves the corollary. □

We close this section with some applications of our results to known transformations.

a) The binary transformation ([1,4]) represents a set S of $n=\Sigma a_i 2^i$, $a_i \in \{o,1\}$, elements by a_i sets of size 2^i for $i \geq 0$. It achieves update and query penalty factor $O(\log n)$.

Corollary 2 (Optimality of the binary transform). *If P is any transformation with width (query penalty factor) $k(n)=O(\log n)$ then the normalized cost (update penalty factor) $h(n)=\Omega(\log n)$.*

Proof. It is simple to show that $k(n)=O(\log n)$ implies $\hat{h}(n)=\Omega(\log n)$ where $\hat{h}$ is defined as in corollary 1. □

Corollary 2 generalizes a result of Saxe/Bentley who proved optimality in the class of arboreal transformations.

b) The k-binomial transformation ([4]) represents a set S of

$$n=\sum_{i=1}^{k}\binom{d_i}{i},\ d_k > d_{k-1} > \cdots > d_1 \geq 0,$$

elements by k sets of size $\binom{d_i}{i}$, $1 \leq i \leq k$, respectively. It has query penalty factor k and update penalty factor $(k!n)^{1/k} \approx (k/e)n^{1/k}$, where $e=2.71\ldots$

Corollary 3 (Optimality of the k-binomial transform). *Let P be any transformation with width $k(n) \leq k$. Then the normalized cost $h(n)$ of P satisfies $h(n) \geq (2/4)(k/(k+1))(k!)^{1/k}n^{1/k} - o(n^{1/k})$.*

Proof. In order to get the large constant we need to take a closer look at the proof of corollary 1. It was shown there that $h(n) \geq (2/4)\cdot(k/(k+1))\cdot h$ where h is such that $\binom{h+1+k}{k} \leq n < \binom{h+2+k}{k}$. It remains to show that $h \geq (k!n)^{1/k} - o(n^{1/k})$. Assume otherwise. Then there is some $\varepsilon > 0$ such that for arbitrarily large n $h/(k!n)^{1/k} \leq 1-\varepsilon$ and hence

$$n < \binom{h+2+k}{k} \leq \frac{(h+2+k)^k}{k!} \leq \frac{\left[k+2+(1-\varepsilon)(k!n)^{1/k}\right]^k}{k!}$$

$$\leq \frac{\left[(k!n)^{1/k}\right]^k}{k!} = n,$$

a contradiction □

Corollary 3 extends a result of Bentley/Saxe who proved optimality in the class of arboreal transforms.

c) More generally, Mehlhorn/Overmars ([2]) describe for every smooth function $k(n)$ a transformation P with width (query penalty factor) $O(k(n))$ and normalized cost (update penalty factor) $O(\hat{h}(n))$ where $\hat{h}(n)$ is defined as in corollary 1. This shows that the bound in corollary 1 and hence in theorem 1 is optimal up to constant factors.

Proof of Theorem 1

The proof of theorem 1 proceeds in several steps. In section III.1 we introduce normal form programs and show that in the transformation to normal form programs the efficiency does not seriously deteriorate. In section III.2 we relate the cost and width of normal form programs to properties of trees and binary trees. Finally, in section III.3 we solve the problem in terms of binary trees.

The ideas in section III.1 are original with the author, section III.2 is similar, but more involved than the corresponding claims in Bentley/Saxe and section III.3 was essentially done in Bentley/Saxe. We believe that our organisation of the material in section III.3 is simpler than theirs.

Standard Form Programs

The transformation to standard form programs is in three steps via simple and simple, disjoint programs. We first state the necessary definitions and lemmas and then give the proofs of the lemmas.

Definition. A program is *simple* if

a) every instruction of type II is of the form construct $S_1,\ldots,S_h$ from $T_1,\ldots,T_k$, delete $T_1,\ldots,T_k$

b) in every instruction of type II either $h=1$ or $k=1$

c) instructions of type III are not used.

Lemma 1. *Let P be admissible for n. Then there is a simple program P'' which is admissible for n such that $c(P'')\leq 4\cdot c(P)$ and $w(P'')\leq w(P)$.*

Definition. A program $P=I_1,\ldots,I_m$ is *disjoint* if for all t, $1\leq t\leq m$, the sets which exist after execution of I_t are pairwise disjoint.

Lemma 2. *Let P be a simple program which is admissible for n. Then there is a program P' such that P' is simple, disjoint, admissible for n and $c(P')\leq c(P)$ and $w(P')\leq w(P)$.*

Definition. A program P is in standard form, if it is simple, disjoint and $h=1$ in all instructions of type II.

Remark. Standard form programs are still more general than arboreal programs. They satisfy condition a) of the definition of arboreal transformation, but they do not necessarily satisfy condition b) and c) of that definition.

Lemma 3. *Let P be simple, disjoint and admissible for n. Then there is a P'' such that P'' is in standard form, is admissible for n and $c(P'')\leq c(P)$ and $w(P'')\leq w(P)$.*

Lemma 4. *Let P be admissible for n. Then there is P' in standard form and admissible for n such that $c(P')\leq 4\cdot c(P)$ and $w(P')\leq w(P)$.*

Proof. Immediately from lemmas 1 to 3. □

It remains to prove lemmas 1 to 3.

Proof of Lemma 1. Consider any instruction I_t of type II in P, say

construct $S_1,\ldots,S_h$ from $T_1,\ldots,T_k$ delete $T_{i_1},\ldots,T_{i_l}$.

We replace instruction I_t by the following two instructions; here $T=T_1\cup T_2\cup\cdots\cup T_k$ and

$$\{j_1,\ldots,j_p\}=\{1,\ldots,k\}\setminus\{i_1,\ldots,i_l\}.$$

Construct T from $T_1,\ldots,T_k$, delete $T_1,\ldots,T_k$

construct $S_1,\ldots,S_h,T_{j_1},\ldots,T_{j_p}$ from T, delete T.

Let P' be the program obtained in this way. Then $c(P')\leq 4\cdot c(P)$, $w(P')\leq w(P)$ and P' is admissible for n. P' satisfies condition a) and b) of the definition of simplicity. This implies, that whenever a set is used in an instruction of type II and III it is also deleted. Hence a set is used at most once is such an instruction. We obtain P'' from P' as follows. Consider any instruction I_t of type III in P', say "delete T". Consider the largest $t'<t$ with: $I_{t'}$ is of type II and constructs T ($I_{t'}$ cannot be of type I because P is admissible). We delete T from the set constructed by $I_{t'}$, and we delete instruction I_t from P. Obviously $c(P'')\leq c(P')$ and $w(P'')\leq w(P')$. Since T is used in instruction I_t of P' it is never used in an instruction of type II in P'; thus the elimination is allowed and P'' is admissible. □

Proof of Lemma 2. Let $P=I_1,\ldots,I_m$ be a simple program. If P is not disjoint then there exists a maximal t such that I_t is of the form

construct $S_1,\ldots,S_h$ from S, delete S

such that $S_1,\ldots,S_h$ are not pairwise disjoint.

Let $S,R_1,\ldots,R_l$ be the sets which exist prior to execution of instruction I_t. For every $i\in S$ one of the following two cases applies.

Case 1. Replacing S_j by $S_j'=S_j-\{i\}$ for all j, $1\leq j\leq h$, in instruction I_t and in future references (there is at most one) to set S_j will not destroy admissibility. In this case we carry out this replacement. It will increase neither cost nor width.

Case 2. Replacing S_j by $S_j'=S_j-\{i\}$ for all j, $1\leq j\leq h$, destroys admissibility. Then there exists j_0, $j_0\in\{1,\ldots,h\}$, such that replacing S_j by S_j' for all j, $j\neq j_0$, will not destroy admissibility. j_0 can be found by working backwards through program P. After execution of P there is at least one set which contains i. Choose one of them, say R. Consider the instruction which produces R, say $I_{t'}$. If $t'<t$ then we are in case 1, if $t'=t$ then $R=S_{j_0}$. If $t'>t$ then one of the sets deleted by instruction I_t, contains i. Let R be one of these sets and continue.

Replacing S_j by S_j' for $j \neq j_0$ will increase neither width nor cost, nor will it destroy admissibility.

In either case we increased the "disjointness" of our program without increasing cost or width. An induction argument finishes the proof of lemma 2. □

Proof of Lemma 3. Let $P = I_1, \ldots, I_m$ be simple, disjoint and admissible for n. If P is not in standard form then there is a maximal t such that I_t is

construct $S_1, \ldots, S_h$ from S, delete S

with $h \geq 2$. Since P is disjoint $S_1, \ldots, S_h$ are a partition of S ($S_1, \ldots, S_h$ must be pairwise disjoint by disjointness of P, their union must be S by admissibility).

We define the depth of a set R with respect to a standard form program P' as follows

$$\text{depth}(R) = \begin{cases} 0 & \text{if } R \text{ exists after execution of } P' \\ 1 + \text{depth}(R') & \text{if } R \text{ is destroyed by instruction } I_s \text{ of } P' \text{ and } I_s \text{ produces } R' \end{cases}$$

Since $P' = I_{t+1} \ldots I_m$ is in standard form, depth (S_j) is defined for all j, $1 \leq j \leq h$. Let $j_0, j_0 \in \{1, \ldots, h\}$ be such that depth $(S_{j_0}) \leq$ depth (S_j) for all j, $1 \leq j \leq h$. We obtain P'' by deleting instruction I_t, deleting all references to S_j, $j \neq j_0$, and replacing the (possible) reference to S_{j_0} by a reference to S.

Apparently, $w(P'') \leq w(P)$. Next we have to show that $c(P'') \leq c(P)$. For i, $i \in \{1, \ldots, n\}$, let $s(i)$ be the number of times element i changes sets during execution of P'. Since P is disjoint, $s(i)$ is well defined. Then $c(P') = 2 \cdot \sum_{i=1}^{n} s(i)$. From depth $(S_{j_0}) \leq$ depth (S_j) for all $j \neq j_0$, we conclude $s'(i) \leq s(i)$ where s' is defined with respect to instructions $I_{t+1}, \ldots, I_m$ of P''. Hence $c(P'') \leq c(P)$. An induction argument finishes the proof. □

Standard Form Programs and Trees

In this section we associate ordered forests with standard form programs and express the cost and width of a standard form program in terms of the associated trees.

Let $P = I_1, \ldots, I_m$ be a standard form program which is admissible for n. We associate an *ordered* forest $F(P)$ with n leaves and $m-n$ internal nodes with P. For every i, $1 \leq i \leq n$, we have a leaf. These leaves also represent the n instructions of type I. The $m-n$ internal nodes represent the $m-n$ instructions of type II. Consider any instruction I_t of type II. Since P is in standard form, I_t is of the form

construct $S_1 \cup S_2 \cup \cdots \cup S_k$ from $S_1, \ldots, S_k$, delete $S_1, \ldots, S_k$.

We may assume w.l.o.g. that $\min S_1 < \min S_2 < \cdots < \min S_k$. Let I_{t_i} be the instruction which produces set S_i, $1 \le i \le k$. Then $v(I_t)$, the node corresponding to instruction I_t, has sons $v(I_{t_1}), \ldots, v(I_{t_k})$ in that order from left to right. Finally, if r sets exist after execution of program P then forest $F(P)$ has r roots. Again we order the roots according to minimal elements in the sets represented by them.

Definition. With every node $v(I_t)$ of forest $F(P)$ we associate its lifespan $[t, u]$, where either $u=m$ and $v(I_t)$ is a root of $F(P)$ or $v(I_t)$ is not a root and $v(I_{u+1})$ is the father of $v(I_t)$.

For node v let $s(v)$ be the depth of v in forest $F(P)$, the depth of a root being zero.

Lemma 5.

a) $c(P) = 2 \cdot \sum_{v \text{ leaf of } F(P)} s(v)$ $(= 2 \cdot \text{external pathlength})$

b) $w(P) = \max_{1 \le t \le m} |\{v; t \text{ belongs to the lifespan of } v\}|; \ 1 \le t \le m$

Proof. Obvious. □

At this point, we almost arrived at a manageable representation of the problem. We related the cost of P with the external path length of a forest and we related the width of P with the "width" of forest P. The second quantity becomes more managable if we consider the standard way of representing an ordered forest as a binary tree.

Definition.

a) a tree T is a *Binary Tree* if

a1) the edges of T are labelled by L and R

a2) every node of T has outdegree ≤ 2. If node v has outdegree 2 then the outgoing edges are labelled differently.

b) Let T be a Binary Tree and let v be a node of T. We define:

b1) $|T|$ = number of nodes of T without a left son,

b2) $ld(v)(rd(v))$ = number of edges labelled $L(R)$ on the path from the root to v,

b3) $Ld(T) = \Sigma\{ld(v); v \text{ is a node of } T \text{ and has no left son}\}$, and

b4) $rd(T) = \max\{rd(v); v \text{ is a node of } T\}$.

Let P be a standard form program which is admissible for n. Let $F(P)$ be the ordered forest associated with it. $F(P)$ has n leaves and $m-n$ internal nodes. We will construct a Binary Tree $T(P)$ with m nodes as follows. If v is a node (leaf or internal node) of $F(P)$ then $\hat{v}$ denotes the corresponding node of $T(P)$.

1) Let v be an internal node of $F(P)$ with sons $v_1, \ldots, v_k$ in that order. Then in $T(P)$ there is a left edge from $\hat{v}$ to $\hat{v}_1$ and there are right edges from $\hat{v}_i$ to $\hat{v}_{i+1}$, $1 \le i < k$.

2) Let $r_1, \ldots, r_p$ be the roots of $F(P)$ in order. Then there are right edges from $\hat{r}_i$ to $\hat{r}_{i+1}$ for $1 \le i < p$.

Lemma 6.

a) *v is a leaf of $F(P)$* iff *$\hat{v}$ has no left son*

b) $s(v)=ld(\hat{v})$

c) $c(P)= 2\cdot\sum_{v \text{ leaf of } F(P)} s(v) = 2\cdot\sum_{\hat{v} \text{ has no left son}} ld(\hat{v}) = 2\cdot Ld(T(P))$

d) $w(P)\geq \max\{rd(\hat{v});\ \hat{v} \text{ has no left son}\}$
$= rd(T(P))$

Proof. a), b) and c) are obvious.

d) Let $\hat{v}$ be a node of $T(P)$ with $rd(\hat{v})=k=rd(T(P))$. Since $T(P)$ is finite we may assume that $\hat{v}$ has no son. Let $\hat{v}_1,\ldots,\hat{v}_k$ be the nodes which are left via right edges on the path from the root to node $\hat{v}$, let $\hat{v}_{k+1}=\hat{v}$. Let v_i be the node of $F(P)$ which corresponds to $\hat{v}_i$, let I_{t_i} be the instruction represented by v_i and let S_i be the set constructed by I_{t_i}.

Then $S_{k+1}=\{j\}$ for some j. Furthermore, $\min S_1<\min S_2<\cdots<\min S_k<\min S_{k+1}=j$ by definition of $F(P)$ and $T(P)$.

Note first, that t_{k+1} is in the lifespan of node $v_{k+1}=v$.

For i, $1\leq i\leq k$, consider the sequence $\hat{v}_{i,o}=\hat{v}_i, v_{i,1},\ldots$ of nodes which are reachable from $\hat{v}_i$ via left edges. For distinct i's these paths are disjoint. Also, the lifespan of the last node of such a path begins before t_{k+1} since $\min S_i<j$. Hence any such paths contains a node whose lifespan contains t_{k+1}. This shows $w(P)\geq k$. □

Lemma 6 finally reduced our problem to a managable problem about binary trees.

Definition. $L_k(n)=\min\{Ld(T);\ T$ is a finite Binary Tree with $|T|=n$ and $rd(T)\leq k\}$

We summarize Lemmas 1 to 6 in

Lemma 7 (part 1 of theorem 1). *Let P be a program which is admissible for n. Let $k=w(P)$. Then*

$$c(P)\geq (2/4)\cdot L_k(n)$$

Proof. By Lemma 4 there is a standard form program P' admissible for n with $c(P')\leq 4\cdot c(P)$ and $w(P')\leq w(P)$. Let $T(P')$ be the Binary Tree associated with P'. Then $c(P')=2\cdot Ld(T(P'))$ and $rd(T(P'))\leq w(P')$ by Lemma 6. Hence $rd(T(P'))\leq k$ and thus $Ld(T(P'))\geq L_k(n)$ since $|T(P')|=n$ by admissibility of P and P'. This shows that $c(P)\geq 2/4\cdot L_k(n)$. □

Determining $L_k(n)$

In this section we will compute $L_k(n)$ for infinitely many n (Lemma 9) and show that $L_k(n)/n$ is non-decreasing (Lemma 8).

Definition. T is optimal for k and n if $rd(T)\leq k$, $|T|=n$ and $Ld(T)=L_k(n)$.

Lemma 8 (part 3 of theorem 1). *$L_k(n+1)/(n+1) \geq L_k(n)/n$ for all $n \geq 1$.*

Proof. Let T be optimal for $n+1$ and k. Let v be a node of T of maximal left-depth and without right son. Let T' be obtained by deleting v from T. Then $|T'|=n$ since $|T'| \geq |T|-1=n$ is obvious and $|T'|=n+1$ contradicts the optimality of T. Hence v is a right son. Furthermore, $L_k(n) \leq Ld(T')$, $Ld(T)=Ld(T')+1d(v)$ and $Ld(T') \leq n \cdot 1d(v)$. Hence

$$\begin{aligned} L_k(n+1) &= Ld(T) \\ &= Ld(T') + 1d(v) \\ &\geq (1+1/n) Ld(T') \\ &\geq (1+1/n) L_k(n) \end{aligned}$$

□

Next we will compute $L_k(n)$ for "nice" values of n by explicitly defining the optimal tree. In the formulation of lemma 9 we identify a node with the path from the root to that node given as a sequence of L's and R's. Also we use $|w|_L$ to denote the number of L's in string w.

Lemma 9 (part 2 of theorem 1). *Let $h, k \geq 0$ and let $n = \binom{h+1+k}{k}$. Consider tree T whose node set consists of all strings $w \in \{L, R\}^*$ with*

1) $|w|_L \leq h, |w|_R \leq k$
2) if $|w|_R = k$ then $|w| = w'R$

Then T is optimal for k and n and

$$Ld(T) = \frac{h \cdot k}{h+k+1} \binom{h+k+2}{k+1}$$

Proof. a) We show first that $|T|=n$. Call a node terminal if it has no left son. A node w of T is terminal if wL is not a node of T. Hence we must either have $|w|_L = h$ or $|w|_R = k$. In the second case w ends in an R. Hence the set of terminal nodes of T is equal to the union of the set $L_i = \{w;\ |w|_L = i, |w|_R = k, w$ ends in an $R\}$, $0 \leq i \leq h$ and the set $L = \{w;\ |w|_L = h, |w|_R \leq k-1\}$. Hence

$$\begin{aligned} |T| &= \sum_{i=0}^{h} |L_i| + |L| \\ &= \sum_{i=0}^{h} \binom{i+k-1}{k-1} + \sum_{j=0}^{k-1} \binom{h+j}{j} \\ &= \binom{h+k}{k} + \binom{h+k}{k-1} = \binom{h+k+1}{k} \end{aligned}$$

b) Next we compute the left path length of T. We have

$$Ld(T) = \sum_{i=0}^{h} i|L_i| + h \cdot |L|$$

$$= \sum_{i=0}^{h} i\binom{i+k-1}{i} + h \cdot \binom{h+k}{k-1}$$

$$= k \cdot \binom{k+h}{h-1} + h \cdot \binom{h+k}{k-1} = \frac{h \cdot k}{h+k+1}\binom{h+k+2}{k+1}$$

c) It remains to show that T is optimal for k and n. Assume otherwise. Then let T' be optimal for h and n and closest to T, i.e. T' and T share a maximal number of nodes. Let $N(T') \subseteq \{L, R\}^*$ be the set of nodes of T'. If $N(T') \subseteq N(T)$ and $T' \neq T$ then $|T'| < n$, a contradiction. Hence there exists a node $w \in N(T') - N(T)$. Then either $|w|_L > h$ or $|w|_R = k$ and w ends in an L. In the second case w does not have a right son (since $|w|_R = k$) and hence we may identify w with its father without reducing the number of terminal nodes but with a decrease in left path length. Hence every node w of T' with $|w|_R = k$ ends in an R.

So there has to be a w in T' with $|w|_L > h$. Furthermore, there has to be a $w' \in N(T) - N(T')$, since $N(T) \subseteq N(T')$ contradicts the non-optimality of T. We may assume w.l.o.g. that w is a longest word in $N(T') - N(T)$ and $w' \in N(T) - N(T')$ is a shortest word.

Case 1. w' ends in an R. Then obtain T'' by deleting w from T' and adding w'. Then $|T''| = |T'|$ and $Ld(T'') < Ld(T')$.

Case 2. w' ends in an L. Then $rd(w') < k$. Obtain T'' by deleting w from T' and adding w' and $w'R$. Then $|T''| = |T'|$. And $Ld(T'') = Ld(T') - ld(w) + ld(w') + 1 \leq Ld(T')$. Hence T'' is optimal for k and n and is closer to T than T'.

In either case we derived a contradiction.

Lemmas 7, 8 and 9 prove theorem 1. □

References

1. J. L. Bentley, "Decomposable Searching Problems," *Information Processing Letters* 8 (5), 244–251 (1979).
2. K. Mehlhorn and M. H. Overmars, Optimal Dynamization of Decomposable Searching Problems, FB 10, Universität des Saarlandes, Techn. Report No. A 80/15, to appear in Information Processing Letters.
3. M. H. Overmars and v. Leeuwen, Two general methods for dynamizing decomposable searching problems, Techn. Report University of Utrecht, RUU-SC-79-10, Nov. 79.
4. J. L. Bentley and J. B. Saxe, Decomposable Searching Problems I. Static-to-Dynamic Transformation, Journal of Algorithms 1, 301–358 (1980).
5. J. van Leeuwen and D. Wood, Dynamization of Decomposable Searching Problems, Information Processing Letters 10, 51–56 (1980).
6. D. E. Willard, Balanced Forests of *k-d* Trees as a Dynamic Data Structure, Harvard Aiken Computer Lab. Report, TR-23-78.

Received June 1, 1980, and in revised form December 5, 1980, and April 23, 1981.