Optimal Dynamization of Decomposable Searching Problems

by

Kurt Mehlhorn
and
Mark H. Overmars *

Oktober 1980

Kurt Mehlhorn, Fachbereich 10 - Informatik
Universität des Saarlandes
6600 Saarbrücken
West Germany

Mark H. Overmars, Vakgroep informatica,
Rijksuniversiteit Utrecht
3508 TA Utrecht
The Netherlands

Abstract: We describe a general method for dynamizing data structures for decomposable searching problems. Our method generates optimal transformations and in fact yields all optimal transformations.

*) The research of the second author is partly supported by the Netherlands Organization for the Advancement of Pure Research (ZWO)

I. 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 [5] 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 :

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

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

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

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

For instance, a Member query is decomposable ($\square$ = or). Saxe/Bentley [5] and later Overmars/v. Leeuwen [4] presented several general methods for dynamizing static data structures for decomposable searching problems.

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 :

a) a set A is represented by static data structures for some partition A_i, $1 \leq i \leq r$, of A, i.e. $A = \bigcup_{i=1}^{r} A_i$ and $A_i \cap A_j = \emptyset$ for $i \neq j$.

b) a query about set A is answered in two steps : first the data structures are used to obtain the answers for the blocks A_i and then the final answer is obtained by decomposability. The cost for the first step is $\sum_{i=1}^{r} Q_S(|A_i|) \leq r \cdot Q_S(n)$, the cost of the second step is assumed to be dominated by the first step and is therefore ignored.

c) the insertion of a new point x is processed by selecting data structures for some blocks $A_{i_1}, A_{i_2}, \ldots, A_{i_s}$ of the partition of A, throwing away these data structures and constructing a new static data structure for point set $\{x\} \cup A_{i_1} \cup A_{i_2} \cup \ldots \cup A_{i_s}$.

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 partition of set A after the i-th insertion, and let m_i be the size of the set for which a static data structure is constructed after the i-th insertion. 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 characteristices were presented in Saxe/Bentley [5] and Overmars/v. Leeuwen [4] , e.g. the binary transform with characteristic (log n, log n) the k-binomial transform with characteristic $(k, (k!n)^{1/k}), \ldots$.

Lower bounds on the efficiency of transformation were shown in Saxe/Bentley [5] and later generalized in Mehlhorn [2]. Mehlhorn proved the following result.

Theorem (Mehlhorn [2]) : There is a constant $c > 0$ such that for any transformation with characteristic $(k(n), h(n))$:

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

where

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

In this note we will show that the lower bound given in the theorem above is sharp (up to constant factors) and exhibit a general method for obtaining all optimal transforms. More specifically, we will show :

Theorem: 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.

Note, that we excluded the case that h(n) "wiggles" about log n. With some additional effort we could also handle this case. For reasons of simplicity we have refrained from doing so.

II. A general method of dynamization

In this section, $f : \mathbb{N} \to \mathbb{N}$ is any nondecreasing function with $f(i) \geq 2$ for all i.

We describe two general methods of dynamizations, methods A and B. Method A will be used for transformations with low query penalty factor (at most log n) and method B for transformations with small update penalty factor (at most log n). We will describe methods A and B together since they have many similarities. We describe first how a set S of n points is represented and then describe the query and insertion algorithms.

Let S be a set with n points, $n = |S|$. Let i be such that $2^i \leq n < 2^{i+1}$. Then set S is represented as follows.

1) there is a structure S_f containing exactly 2^i points of S.

2) Let $n' = n-2^i$, $b = f(\lfloor \log n \rfloor) = f(i) \geq 2$ and let $n' = \sum_{j \geq 0} a_j b^j$, where $0 \leq a_j < b$, $a_j \in \mathbb{N}$ for all j. Then the remaining n' points of S are stored as follows.

Method A: for each j, $j \geq 0$, there is one structure, called B_j, containing exactly $a_j \cdot b^j$ points of S.

Method B: for each j, $j \geq 0$, there are a_j structures called $B_{j1},\ldots,B_{ja_j}$ containing b^j points each.

This finishes the description of the representations of a set S. Next we describe query and insertion algorithms. The query algorithm is the same for both methods : Answer the query for each of the non-empty structures and derive the final answer by decomposability.

Next consider the insertion of a new point p into a set S, $|S| = n$ and $2^i \leq n < 2^{i+1}$.

If $n+1 = 2^{i+1}$ then we throw away all existing structures and construct a new S_f containing all n+1 points inserted so far.

If $n+1 < 2^{i+1}$ then let j_o be minimal such that $a_{j_o}+1 < b$.

Method A: Throw away structures $B_o,\ldots,B_{j_o}$ and construct a new B_{j_o} containing the new point p and the points contained in the old $B_o,B_1,\ldots,B_{j_o}$. Note that the new B_{j_o} contains

$$1 + \sum_{j<j_o} (b-1)b^j + a_{j_o}\cdot b^{j_o} = (a_{j_o} + 1)\cdot b^{j_o} \text{ points.}$$

Method B: Throw away structures $B_{j,\ell}$ for $0 \le j < j_o$ and $1 \le \ell \le b-1$ and construct a new $B_{j_o,a_{j_o}+1}$ out of the new point p and the points in the $B_{j,\ell}$'s $0 \le j < j_o$, $1 \le \ell \le b-1$. Note that $B_{j_o,a_{j_o}+1}$ contains $1 + \sum_{j<j_o} (b-1)b^j = b^{j_o}$ points.

Theorem 1: Let $f : \mathbb{N} \to \mathbb{N}$ be any nondecreasing function with $f(i) \ge 2$ for all i. Then method X, X = A,B, based on f, yields a transformation with characteristic

$$(1 + g_1(n),\ 2 + g_2(n)) \qquad \text{for } X = A$$
$$(1 + g_2(n),\ 2 + g_1(n)) \qquad \text{for } X = B, \text{ where}$$

$g_1(n) = \max\{1 + \lfloor \log(m/2)/\log f(\lfloor \log m \rfloor) \rfloor \ ;\ m \le n\}$
and

$g_2(n) = g_1(n)\cdot(f(\lfloor \log n \rfloor) - 1)$

proof: Consider any n. Let i be such that $2^i \le n < 2^{i+1}$.

We will determine $k^X(n)$ first, where $k^X(n)$ is the query penalty factor of method X, X = A,B.

In order to do so, we need a bound on the maximal number of structures existing at any one time during the first n insertions. Let $m \le n$. If method A is used and B_j (method B is used and B_{j1}) is non-empty after the m-th insertion then $f(\lfloor \log m \rfloor)^j < m/2$ since a non-empty B_j ($B_{j,1}$) contains at least $f(\lfloor \log m \rfloor)^j$

elements, when total set size is m, and at least half of the points are always contained in S_f.

This shows that the maximal number of structures existing at any one time during the first n insertions is

$$\leq 1 + \max_{m \leq n} \left[1 + \left(\frac{\log m/2}{\lfloor \log f(\lfloor \log m \rfloor) \rfloor}\right)\right]$$

if method A is used (the additional one comes from the structure S_f) and

$$\leq 1 + \max_{m \leq n} \left[\left(1 + \left(\frac{\log m/2}{\lfloor \log f(\lfloor \log m \rfloor) \rfloor}\right)\right) \cdot (f(\lfloor \log m \rfloor) - 1)\right]$$

if method B is used. Since f is assumed to be nondecreasing the bound follows.

Next we analyse $h^B(n)$, the update penalty factor of method B. We consider the cost of inserting points 2^i, $2^i+1,\ldots,n$ first. (Here and in the sequel we will frequently abuse the language. Instead of using x_t for the t-th point inserted we will simply use t). When point 2^i is inserted a new S_f of size 2^i is constructed. Each point x, $2^i + 1 \leq x \leq n$, is inserted into at most one $B_{j,\ell}$ for some ℓ and each $j \geq 0$. Hence at most $\lfloor((n-2^i)/f(\lfloor \log n \rfloor)^j)\rfloor$ different $B_{j\ell}$'s, each of size $f(\lfloor \log n \rfloor)^j$, are built during the insertion of points $2^i+1,\ldots,n$.

Thus the total size of all blocks built during the first n insertions is bounded by

$$\sum_{\ell=0}^{i} 2^\ell + \sum_{\ell=0}^{i-1} \sum_{j \geq 0} \left\lfloor \frac{2^{\ell+1}-1-2^\ell}{f(\ell)^j} \right\rfloor f(\ell)^j$$

$$+ \sum_{j \geq 0} \left\lfloor \frac{n-2^i}{f(i)^j} \right\rfloor f(i)^j$$

Using the fact that $\lfloor(2^{\ell+1}-1-2^\ell)/f(\ell)^j\rfloor$ (and $\lfloor(n-2^i)/f(i)^j\rfloor$) vanishes for $j \geq g_1(n)$ we can bound this sum by

$$2\cdot 2^i + \sum_{j=o}^{g_1(n)-1} \left[\sum_{\ell=o}^{i-1} (2^{\ell+1}-1-2^\ell) + (n-2^i) \right]$$

$$\leq 2\cdot n + n\cdot g_1(n).$$

Hence $h^B(n) \leq 2 + g_1(n)$.

Finally we derive a bound on $h^A(n)$. As above, we consider the cost of inserting points $2^i, 2^i+1, \ldots, n$ first. Point 2^i forces us to construct a structure of size 2^i. Each point x, $2^i < x \leq n$, is member of at most one B_j of size $a\cdot f(\lfloor \log n \rfloor)^j$ for each a, $1 \leq a \leq f(\lfloor \log n \rfloor) - 1$ and each $j \geq o$. Hence at most $\lfloor (n-2^i)/(a\cdot f(\lfloor \log n \rfloor)^j) \rfloor$ distinct sets of size $a\cdot f(\lfloor \log n \rfloor)^j$ are formed during the insertion of these points, $j \geq o$, $1 \leq a \leq f(\lfloor \log n \rfloor) - 1$.

Thus the total size of all blocks formed during the first n insertion is bounded by

$$\sum_{\ell=o}^{i} 2^\ell + \sum_{\ell=o}^{i-1} \sum_{j\geq o} \sum_{a=1}^{f(\ell)-1} \left\lfloor \frac{2^{\ell+1}-1-2^\ell}{a\cdot f(\ell)^j} \right\rfloor a\cdot f(\ell)^j$$

$$+ \sum_{j\geq o} \sum_{a=1}^{f(i)-1} \left\lfloor \frac{n-2^i}{a\cdot f(i)^j} \right\rfloor \cdot a \cdot f(i)^j$$

As above, one concludes that this sum is bounded by $2n + n\cdot g_1(n)\cdot(f(\lfloor \log n \rfloor)-1)$. Hence $h^A(n) \leq 2 + g_2(n)$.

□

Remark 1: Observe the duality of parts a) and b) of theorem 1.

Remark 2: if $(i-1)/\log f(i)$ is non-decreasing in i, then

$$g_1(n) = 1 + \max\left\{ \left\lfloor \frac{\log (n/2)}{\log f(\lfloor \log n \rfloor)} \right\rfloor , \frac{\log((2^{\lfloor \log n \rfloor}-1)/2)}{\log f(\lfloor \log n \rfloor -1)} \right\}$$

and

$$g_2(n) = g_1(n)\cdot(f(\lfloor \log n \rfloor) -1)$$

Examples: The following table lists the characteristic of some transformations obtainable by our method.

	f(i)	$k^A(n)$	$h^A(n)$	$k^B(n)$	$h^B(n)$
a)	2	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$
b)	$\lfloor \sqrt[k]{2^i} \rfloor$ for some constant k	$O(k)$	$O(k\cdot\sqrt[k]{n})$	$O(k\cdot\sqrt[k]{n})$	$O(k)$
c)	i	$O(\frac{\log n}{\log\log n})$	$O(\frac{(\log n)^2}{\log\log n})$	$O(\frac{(\log n)^2}{\log\log n})$	$O(\frac{\log n}{\log\log n})$
d)	2^i	$O(1)$	$O(n)$	$O(n)$	$O(1)$
e)	$2^{\sqrt{i}}$	$O(\sqrt{\log n})$	$O(2^{\sqrt{\log n}}\sqrt{\log n})$	$O(2^{\sqrt{\log n}}\sqrt{\log n})$	$O(\sqrt{\log n})$

Note the similartiy between a) and Bentley [1] , b) and Saxe/Bentley [5], and a), c), d), e) and the examples in Overmars/v. Leeuwen [4].

Next we turn to the question of optimality. The following theorem was shown in Mehlhorn [2].

Theorem 2 (Mehlhorn [2]): There is a constant $c > 0$ such that for every transform with characteristic $(k(n), h(n))$: $h(n) \geq c\cdot\hat{h}(n)$ for all n where

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

Theorem 2 gives lower bounds on the efficiency of any transform. In theorem 3 below we show that methods A and B work at these limits and indeed yield all optimal transforms.

Theorem 3: Let T be any transform with characteristic (k(n),h(n)) for some nondecreasing function k(n).

a) if $(i-1)/k(2^i)$ is nondecreasing in i then there is a function f such that transform A based on f has characteristic (O(k(n)), O(h(n))).

b) if k(n)/log n is non-decreasing then there is a function f such that transform B based on f has characteristic (O(k(n)), O(h(n))).

proof: a) since k(n) = O(log n) we infer from theorem 2 $h(n) \geq c \cdot k(n)\, n^{1/k(n)}$ for some c > 0 and all n. Let

$$f(i) = \max(2,\ 2^{\lfloor (i-1)/k(2^i) \rfloor})$$

for all i. Then $f(i) \geq 2$ and f is non-decreasing. Hence transform A based on f has characteristic $(1 + g_1(n),\ 2 + g_2(n))$ where g_1 and g_2 are defined as in theorem 1. Also

$$1 + g_1(n) = 2 + \max\left\{\frac{\log (m/2)}{\log f(\lfloor \log m \rfloor)} \ ;\ m \leq n\right\}$$

$$= 2 + \max\left\{\frac{\log(m/2)}{\max(1, \lfloor (\lfloor \log m \rfloor - 1)/k(2^{\lfloor \log m \rfloor}) \rfloor)} ;\ m \leq n\right\}$$

$$= O\left(\max\left\{\frac{\log(m/2) \cdot k(2^{\lfloor \log m \rfloor})}{(\lfloor \log m \rfloor - 1)} \ ;\ m \leq n\right\}\right)$$

$$= O(k(n))$$

and

$$
\begin{aligned}
2 + g_2(n) &= O(k(n) \cdot f(\lfloor \log n \rfloor)) \\
&= O(k(n) \cdot 2^{(\lfloor \log n \rfloor - 1)/k(2^{\lfloor \log n \rfloor})}) \\
&= O(k(n) \cdot 2^{\lfloor \log n \rfloor / k(2^{\lfloor \log n \rfloor + 1})}) \\
&= O(k(n) \cdot n^{1/k(2^{\lfloor \log n \rfloor + 1})}) \\
&= O(k(n) \cdot n^{1/k(n)}) \\
&= O(h(n))
\end{aligned}
$$

b) since $k(n) = \Omega(\log n)$ we infer from theorem 2 $h(n) = \Omega(\log n / \max(1, \log(k(n)/\log n)))$ for some constant c and all n. Let $f : \mathbb{N} \to \mathbb{N}$ be such that

$$\frac{k(2^i)}{i-1} < \frac{f(i)-1}{\log f(i)} \leq \frac{k(2^i)}{i-1} + 1$$

for all i. Then $f(i) \geq 2$ and f is nondecreasing. Then transform B based on f has characteristic $(1 + g_2(n), 2 + g_1(n))$ where g_1, g_2 are defined as in theorem 1. Apparently, $1 + g_2(n) = O(k(n))$ Also

$$2 + g_1(n) \leq 3 + \max\left\{\frac{\log m/2}{\log f(\lfloor \log m \rfloor)} \; ; \; m \leq n\right\}$$

$$\leq 3 + \frac{\log n}{\log[k(n)/\log n]} = O(h(n))$$

since $\log f(i) = \Omega(\log[(f(i) - 1)/\log f(i)]) =$

$\Omega(\log(k(2^i)/(i-1))) = \Omega(\log(k(2^i)/i)) = \Omega(\log(k(n)/\log n))$ for $i \leq \log n$.

□

Theorem 3 shows that methods A and B suffice to generate all optimal transforms with query penalty factor $k(n) = O(\log n) \cup \Omega(\log n)$ and $k(n)/\log n$ sufficiently smooth. It thus demonstrates the generality of the methods.

If $k(n)$ is not in $O(\log n) \cup \Omega(\log n)$, then a combination of methods A and B can still lead to optimal transforms. One chooses a switchpoint function, say $2 \log n$. As long as $k(n) \geq 2 \log n$ one operates with method B. As soon as $k(n)$ becomes less that $2 \log n$, one leaves the data structures constructed by method B as they are and switches over to method A. Once $k(n)$ becomes larger than $2 \log n$ again, we switch back to method B, i.e. we process the sequence of insertions which were done in mode A first and are then ready for new queries and insertions. If $k(n)/\log n$ is sufficiently smooth then this method will have characteristic $O(k(n)), O(\hat{h}(n)))$ where $\hat{h}$ is defined as in theorem 2. The details are left to the reader.

We want to close with a remark about deletions.
Known deletion methods (Saxe/Bentley, v. Leeuwen/Maurer, Overmars/v. Leeuwen) can be carried over, though the details still need to be done.

Bibliography

[1] Bentley, J.L. : Decomposable Searching Problem, Information Processing Letters, 8(5), pp. 244-251, 1979

[2] Mehlhorn, K. : Lower Bounds on the Efficiency of Static to Dynamic Transformations of Data Structures Techn. Report, A 80/05, FB 10, Universität des Saarlandes 1980

[3] Overmars, M.H, v. Leeuwen, J. : Two general methods for dynamizing decomposable searching problems, to appear in COMPUTING

[4] Overmars, M.H., v. Leeuwen, J. : Some principles for dynamizing decomposable searching problems, Techn. Report RUU-CS-80-1, Dept. of Computer Science, University of Utrecht, 1980 , to appear in Information Processing Letters

[5] Saxe, J.B., Bentley, J.L. : Transforming Static Data Structures to Dynamic Data Structures, 20th IEEE Symposium on Foundations of Computer Science, 1979, pp. 148-168

[6] van Leeuwen, J., Maurer, H.A. : Dynamic Systems of Data Structures, Techn. Report, Institut für Informationsverarbeitung, TU Graz, Austria, 1980