SOME PRINCIPLES FOR DYNAMIZING DECOMPOSABLE SEARCHING PROBLEMS

Mark H. Overmars and Jan van Leeuwen

RUU-CS-80-1

Januari 1980

Rijksuniversiteit Utrecht

Vakgroep Informatica

Budapestlaan 6
Postbus 80.012
3508 TA Utrecht
Telefoon 030–531454
The Netherlands

SOME PRINCIPLES FOR DYNAMIZING DECOMPOSABLE SEARCHING PROBLEMS

Mark H. Overmars and Jan van Leeuwen

Technical Report RUU-CS-80-1

Januari 1980

Department of Computer Science
University of Utrecht
P.O. Box 80.012, 3508 TA Utrecht
the Netherlands

SOME PRINCIPLES FOR DYNAMIZING DECOMPOSABLE SEARCHING PROBLEMS

Mark H. Overmars and Jan van Leeuwen

Department of Computer Science, University of Utrecht
P.O. Box 80.012, 3508 TA Utrecht, the Netherlands

Abstract: We define two principles to obtain dynamizations of decomposable searching problems, both based on a general formalism for number systems. The first class enables us to obtain low insertion times, while the query-time stays reasonable, and the second class enables us to obtain low query-times, while keeping the update time reasonable. We also give a general deletion technique for these methods.

Keywords and phrases: decomposable searching problems, dynamization, number systems.

1. Introduction.

In the past two years, much effort was made to obtain general dynamization methods for classes of problems. Bentley [1], and later Saxe and Bentley [4], defined the class of so-called decomposable searching problems, for which such general techniques indeed exist. A searching problem is a problem in which a question (query) is asked about an object of type T^1, with respect to a set of points of type T^2, with an answer of type T^3.

Definition 1.1. A searching problem is called decomposable if and only if

$$\forall\, x \in T^1,\ A, B \in 2^{T^2} \qquad Q(x, A \cup B) = \square\,(Q(x, A), Q(x, B))$$

for some efficiently computable operator $\square$ on the elements of T^3.

For a decomposable searching problem we may split a set of points into subsets, and can derive the answer to a query from the answers to the same query on the subsets without much additional cost. Saxe and Bentley [4] gave several examples of general dynamization methods, based on different ways of partitioning a set. Typically a set is partitioned in different size subsets and each subset is represented as a static structure (a block). When a new point is inserted, a new structure for it is initiated, or perhaps a small block is rebuilt with the new point included.

After sufficiently many insertions, a number of small blocks are taken together and are rebuilt into one single, larger block. The cost for this "rebuilding" can be averaged over the preceding insertions and will not be excessive, as rebuildings of larger blocks will occur less often. All these methods are based on a way of counting in a number system, with blocks instead of digits.

We will define two main principles for obtaining dynamization methods, based on a general formalism for counting, to obtain a more uniform theory. In section 2 we shall describe the formalism for number systems we need and give some examples of known counting techniques that belong to it. In section 3 we shall define two general classes of dynamization methods based on this frame work. One class permits us to lower insertion times arbitrarily, and the other permits us to lower query times arbitrarily. We also give some examples of specific dynamization methods that belong to the classes. The methods of section 3 are only able to handle insertions. In section 4 we will show how, for a relevant subclass of decomposable searching problems, known deletion techniques apply here just as well.

Throughout this paper we will use the following notation:

$Q_S(N)$= the query time for a static structure with N points
$P_S(N)$= the time required to build a static structure of N points
$Q_D(N)$= the query time for a dynamic structure of N points
$I_D(N)$= the average insertion time over N transactions on an initially empty structure.
$D_D(N)$= the average deletion time over N transactions on an initially empty structure (when applicable).

We assume that Q_S is nondecreasing and that $\frac{P_S(i)}{i} \leq \frac{P_S(j)}{j}$ for $i \leq j$.

2. A general way of counting.

Most number systems are based on two sequences $a=\{a_j\}_{j\geq 1}$ and $b=\{b_j\}_{j\geq 1}$ of positive integers, with a increasing, such that every number $n \geq 0$ can be written as

$$n = \sum_{j>0} n_j a_j \quad \text{with} \quad 0 \leq n_j \leq b_j$$

in at least one way.

For instance, in the decimal number system one would have: $a_j=10^{j-1}$ and $b_j=9$. The n_j are just the "digits" in the particular number system that is chosen. We would like number representations to be unique, i.e., such that every n can be written in exactly one way as $\sum_{j\geq 0} n_j a_j$.

We recall a wellknown fact about mixed radix number systems (cf. Knuth [2], p 175).

Theorem 2.1. A number representation is unique and complete if and only if $a_i = \prod_{j=0}^{i-1} (b_j+1)$ (provided we define $b_0 \equiv 0$) for all $i \geq 1$.

It follows that a number system is completely determined by the sequence b only. Observe that $a_i = \prod_{j=0}^{i-1} (b_j+1)$ implies that $a_i = \sum_{j=1}^{i-1} (b_j a_j) + 1$.

By choosing different sequences b one can obtain many different number systems.

Examples:

a) $b_i=1$ for all $i > o$. In this case $a_1=1$ and $a_i = \prod_{j=1}^{i-1} 2 = 2^{i-1}$.

Hence the number system is the wellknown binary number system.

b) $b_i=k-1$ for all $i > 0$. It follows that $a_i = \prod_{j=0}^{i-1} (b_j+1) = \prod_{j=1}^{i-1} k = k^{i-1}$.

Hence we get the k-ary number system.

c) $b_i=i$ for all i. This is a more interesting example . It has $a_i = \prod_{j=0}^{i-1} (j+1) = i!$

It is the so called factorial number system (cf. Knuth [2] ,p.175).

d) $b_1 = \infty$. In this case $a_1=1$ and the a_i $(i>1)$ are "undefined", and every n can be written as $n=n_1.1$ .The number system is the familiar unary number system.

3. Two classes of dynamization methods.

In this section we will use the general framework of counting, described in section 2, to obtain two classes of dynamization methods for decomposable searching problems. We shall only consider insertions, for the time being. The first class is constructed in such a way that low insertion times are achieved.

Let b be the defining sequence of a number system and let $a_i = \prod_{j=0}^{i-1} (b_j+1)$.

The dynamic system we will use consists of "bags" BA_1, BA_2, ... in which BA_i contains at most b_i static structures B_i (called blocks) of size a_i. Recall that every number n can be written in exactly one way as $n=\sum_j n_j a_j$ with $n_j \leq b_j$. So for every n there is exactly one structure that contains n points, namely the structure in which every BA_j contains just n_j blocks ($j \geq 1$). See figure 1.

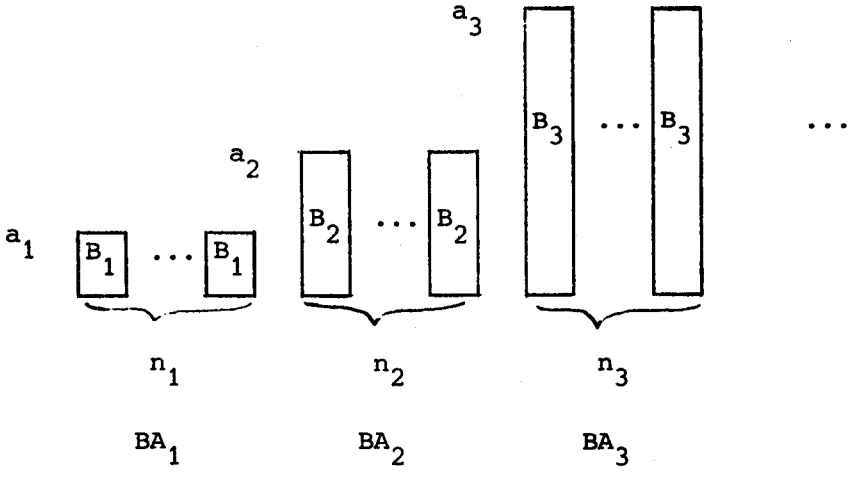

Figure 1

We will call a bag BA_i full if it contains b_i blocks and we will call it empty if it contains no blocks. When we want to insert a point in a dynamic structure of n points, we will have to change it into the structure of n+1 points. When we look at the number system, we see that adding 1 means a) finding the first n_j with $n_j < b_j$, b) setting $n_1 \ldots n_{j-1}$ to 0 and c) adding 1 to n_j. When we want to insert a point p into the structure, we will proceed in a very similar way and first determine the smallest j for which BA_j is not full, empty $BA_1 \ldots BA_{j-1}$ and build a new B_j of the points from $BA_1 \ldots BA_{j-1}$ and p (it are exactly a_j points).

Theorem 3.1. Let $a_i \leq N < a_{i+1}$. The dynamic structure described above has a querytime of $Q_D(N) \leq \sum_{j=1}^{i} b_j Q_S(a_j)$, and an average insertion time of $I_D(N) \leq \sum_{j=1}^{i} P_S(a_j)/a_j$.

Proof.

Let us first consider the querytime. Because $a_i \leq N < a_{i+1}$, BA_j must be empty for all $j > i$. Every BA_j $(j \leq i)$ contains at most b_j blocks of size a_j each. When we want to perform a query over BA_j we have to perform the query on every block in BA_j, which takes at most $b_j\, Q_S(a_j)$. Hence, a query over the whole structure takes at most $\sum_{j=1}^{i} b_j Q_S(a_j)$.

To obtain a bound for the average insertion time consider what happens when we insert a point p. Most of the time BA_1 will not be full, so we can just build a structure consisting of p only and put it in BA_1 (a_1 is always 1). Sometimes we have to take some bags together and must build a new structure out of their points. Note that in such a case all points are moved into a higher indexed bag. Building a block B_j $(j \leq i)$ takes $P_S(a_j)$. When we divide the cost over the points a block is built from, each point is charged $P_S(a_j)/a_j$ when it moves into a larger block. Since no bag beyond BA_i can be filled, the total charge per point is $\sum_{j=1}^{i} P_S(a_j)/a_j$. Hence the average insertion time is bounded by $\sum_{j=1}^{i} P_S(a_j)/a_j$.

□

One easily verifies

Corollary 3.1. Let $a_i \leq N \leq a_{i+1}$. The dynamic structure described above has a querytime of $Q_D(N) \leq \sum_{j=1}^{i} b_i\, Q_S(N)$, and an average insertion time of $I_D(N) \leq i\, P_S(N)/N$.

Let us look at some examples again.

a) $b_i = 1$ for all $i > 0$. In this case $a_i = 2^{i-1}$ (See section 2). If $a_i \leq N < a_{i+1}$, then $i = O(\log N)$ and $\sum_{j=1}^{i} b_j = i = O(\log N)$. It follows that the dynamization method based on this counting system achieves

$$Q_D(N) \leq O(\log N)\, Q_S(N)$$

$$I_D(N) \leq O(\log N)\, P_S(N)/N$$

It is the binary method of Saxe and Bentley [4].

b) $b_i = k-1$ for $i > 0$. It yields $a_i = k^{i-1}$. If $a_i \le N < a_{i+1}$, then

$i = O(^k\log N)$ and $\sum_{j=1}^{i} b_j = O(k\ ^k\log N)$. Hence

$$Q_D(N) \le O(k\ ^k\log N)\ Q_S(N)$$
$$I_D(N) \le O(\ ^k\log N)\ P_S(N)/N$$

c) $b_i = i$ for all $i \ge 0$. In this case $a_i = i!$. Let $a_i \le N < a_{i+1}$, i.e. $i! \le N < (i+1)!$ One easily verifies from Stirling's formula that necessarily $i = O\left(\frac{\log N}{\log\log N}\right)$. Hence a dynamization based on this system achieves

$$Q_D(N) \le O\left(\left(\frac{\log N}{\log\log N}\right)^2\right) Q_S(N)$$
$$I_D(N) \le O\left(\frac{\log N}{\log\log N}\right) P_S(N)/N$$

d) $b_i = \infty$. In this (degenerated) case one always has $i = 1$. Of course, the number of B_1-structures is always at most N, and in this case it is exactly N. Thus the method achieves (because $a_1 = 1$)

$$Q_D(N) \le O(N)\ Q_S(1)$$
$$I_D(N) \le 1.\ P_S(1)/1 = P_S(1)$$

e) $b_i = 2^i-1$ for all $i \ge 0$. Then $a_i = 2^{i-1} \ldots 2^1.\ 1 = 2^{\frac{1}{2}i(i-1)}$. If

$a_i \le N < a_{i+1}$, then obviously $i \le \sqrt{2\log N}+1$.

Also $\sum_{j=1}^{i} b_j \le O\left(\sum_{j=1}^{\sqrt{2\log N}+1} 2^j-1\right) = O\left(2^{\sqrt{2\log N}}\right)$. Hence

$$Q_D(N) \le O\left(2^{\sqrt{2\log N}}\right)\ Q_S(N)$$

$$I_D(N) \le O\left(\sqrt{\log N}\right) P_S(N)/N$$

There is a relation between the multiplicative factor of $Q_S(N)$ and of $P_S(N)/N$ in theorem 3.1 which is worth noting.

Theorem 3.2. Let $f(N)$ be an increasing function with $f(1) = 1$ and $\lim f = \infty$. Let f^{-1} be the inverse of f. There exists a sequence b such that the dynamization method implied by b achieves an average insertion time of $I_D(N) = O(f(N)\ P_S(N)/N)$, while the querytime remains at

$$Q_D(N) = O(g(N)\ Q_S(N)) \text{ for } g(N) \le \sum_{j=1}^{f(N)} \left(\left\lceil \frac{f^{-1}(j+1)}{f^{-1}(j)} \right\rceil - 1\right).$$

Proof.

Let $b_i = \left\lceil \frac{f^{-1}(i+1)}{f^{-1}(i)} \right\rceil - 1$ for $i > 0$. Then $a_i = \prod_{j=0}^{i-1} (b_j+1) = \prod_{j=0}^{i-1} \left\lceil \frac{f^{-1}(j+1)}{f^{-1}(j)} \right\rceil \geq f^{-1}(i)$.

Hence, if $a_i \leq N < a_{i+1}$, then $f^{-1}(i) \leq N$ and $i \leq f(N)$. It follows that

$$I_D(N) = O(i\ P_S(N)/N) = O(f(N)\ P_S(N)/N)$$

$$Q_D(N) = O\left(\sum_{j=1}^{i} b_j\ Q_S(N) \right) = O\left(\left(\sum_{j=1}^{f(N)} \left\lceil \frac{f^{-1}(j+1)}{f^{-1}(j)} \right\rceil - 1 \right) Q_S(N) \right)$$

□

Note that the estimate on g(N) in theorem 3.2. is rather crude and can become very large for "small" f. One can choose many other b's in the proof of 3.2. and obtain similar results, with sharper estimates perhaps when explicit formulae for f and b are known. Theorem 3.2. shows that the multiplicative factor for $P_S(N)/N$ can be made arbitrarily "small".

Because $b_i \geq 1$ for all i, the multiplicative factor of the querytime in theorem 3.1. is always greater than (or equal to) the multiplicative factor of $P_S(N)/N$. So, to obtain a method with lower querytime we have to use another dynamization. To this end we introduce a second class of methods, again based on the general formalism for number systems described in section 2. We again divide the points over bags BA_1, BA_2, ... but, instead of allowing several blocks in each bag, we now require that each bag BA_i contains just one block, that has size $n_i a_i$ for some integer $0 \leq n_i \leq b_i$. Hence, when $n = \sum_j n_j a_j$, we have a block of size $n_1 a_1$ in BA_1, a block of size $n_2 a_2$ in BA_2, etc. See figure 2.

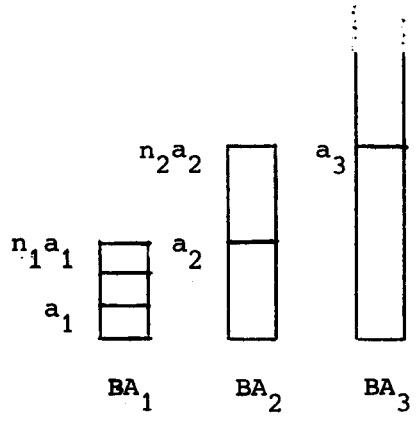

Figure 2

BA_j is called full if it contains a block of size $b_j a_j$ and empty if it contains no points.

Inserting a new point p proceeds in almost the same way as in our first model. We determine the smallest j for which BA_j is not full. We empty all BA_i with $i < j$, but, instead of adding a new block to BA_j like we did before, we rebuild the current block in BA_j together with the points from $BA_1, \ldots, BA_{j-1}$ and p into a single larger block. Because the number of points in $BA_1, \ldots, BA_{j-1}$ and p is exactly a_j and BA_j contained a block with size $n_j a_j$ with $n_j < b_j$, we obtain a new block in BA_j of size $(n_j+1)a_j$ which is legitimate.

Theorem 3.3. Let $a_i \leq N < a_{i+1}$. The dynamic structure described has a querytime of $Q_D(N) \leq i\, Q_S(N)$ and an average insertion time of

$$I_D(N) \leq \sum_{j=1}^{i} b_j\, P_S(N)/N.$$

Proof.

Because $a_i \leq N < a_{i+1}$, BA_j is empty for $j > i$. Hence at most the first i bags can contain points. Every bag contains at most 1 block. The size of every block is at most N so the querytime is at most $i\, Q_S(N)$. To obtain a bound for the average insertion time, we again look at the number of times a point gets built into a structure. The first time p gets in BA_j, it is built into a block of some size $n_j a_j$. After another a_j insertions this block will have to be rebuilt and gets size $(n_j+1)a_j$. This can happen at most b_j times, because when BA_j gets full p is moved into a higher indexed bag. Hence, in every BA_j, a point can be built at most b_j times in a block. Because $a_i \leq N < a_{i+1}$, every point cannot be built more than $\sum_{j=1}^{i} b_j$ times in a block. Dividing the cost for building a block over the points it is built from yields a total charge of at most $\sum_{j=1}^{i} b_j\, P_S(N)/N$ per point. Hence the average insertion time is bounded by $\sum_{j=1}^{i} b_j\, P_S(N)/N$.

□

Note the duality between this theorem and theorem 3.1. (corollary 3.1.). Whe we use this dynamization, it is easy to see what it yields in our examples, since we can just swap the multiplicative factors of Q_D and I_D.

a) $b_i = 1$

$$Q_D(N) = O(\log N)\, Q_S(N)$$
$$I_D(N) = O(\log N)\, P_S(N)/N$$

b) $b_i = k-1$

$$Q_D(N) = O({}^k\log N)\, Q_S(N)$$
$$I_D(N) = O(k\, {}^k\log N)\, P_S(N)/N$$

c) $b_i = i$ $\quad Q_D(N) = O\left(\frac{\log N}{\log \log N}\right) Q_S(N)$

$I_D(N) = \Theta\left(\left(\frac{\log N}{\log \log N}\right)^2\right) P_S(N)/N$

d) $b_i = \infty$ $\quad Q_D(N) = Q_S(N)$

$I_D(N) = O(N)\ P_S(N)/N = \tilde{\Theta}(P_S(N))$

e) $b_i = 2^{i-1}$ $\quad Q_D(N) = O(\sqrt{\log N})\ Q_S(N)$

$I_D(N) = O\left(2^{\sqrt{2 \log N}}\right)\ P_S(N)/N$

Also theorem 3.2. can be adapted to the dual situation.

Theorem 3.4. Let $f(N)$ be an increasing function with $f(1) = 1$ and $\lim f = \infty$ Let $f^{-1}(N)$ be the inverse of $f(N)$. There exists a sequence b such that the dynamization method implied by b (in the second manner) has a querytime of $Q_D(N) \leq O(f(N)\ Q_S(N))$ and an average insertion time of

$I_D(N) \leq O(g(N)\ P_S(N)/N)$ where $g(N) \leq \sum_{j=1}^{f(N)} \left(\left\lceil \frac{f^{-1}(j+1)}{f^{-1}(j)} \right\rceil - 1\right)$.

Proof.

The proof is completely similar to the proof of theorem 3.2.

□

The general counting system we have described does not cover all known number systems. For instance, the binomial counting system, as used by Saxe and Bentley [4], does not fit into this scheme. To accomodate such number systems also, we have to allow the b_i to be a function of both i and N. We intend to investigate this at some later time.

4. Accomodating deletions.

Until now we have only considered insertions. Often one also would like to delete points from a set (and thus from the structure). The condition of decomposability of a problem is too weak to obtain a general structure with both a reasonable querytime and a reasonable average deletion time (cf. Saxe and Bentley [4]), and one has to put some more restrictions on the problems to handle them in a sufficiently general way. A first attempt in this direction was made by Saxe and Bentley [4]. The problems they considered were later renamed into "decomposable counting problems" by van Leeuwen and Maurer [5].

Definition 4.1. A decomposable searching problem is called a decomposable counting problem if and only if for any set of points S and any subset S' of S, the answer Q(S/S') can be synthesized at only nominal extra cost from Q(S) and Q(S').

Given a decomposable counting problems, it is not necessary to perform deletions immediately when they occur. We can buffer them in some way for a while. Saxe and Bentley [4] describe how this property can be used to obtain a deletion strategy for the dynamizations they describe, which is fast on the average. The idea is to implement the buffer as just another dynamic structure of the same kind, as a "ghost-structure", containing all deleted points. Insertions are always performed on the primary structure and are processed as usual. When the size of the ghost-structure becomes half the size of the primary structure, one builds a completely new structure of all points currently left and throw away all deleted points. So the ghost-structure becomes empty again. One can easily see that this technique applies not only to the methods described by Saxe and Bentley [4] but to all methods we have characterized in this paper.

Theorem 4.1. Given a decomposable counting problem and a dynamization method with an average insertion time of $O(f(N)\ P_S(N)/N)$, there is a way to perform deletions in average time $O(f(N)\ P_S(N)/N)$ as well (for $f(N) \geq 1$).

Proof.

We use the method to perform deletions described above. Usually, performing a deletion consists of inserting a point into the ghost-structure. Because the ghost-structure is smaller than the primary structure, the time needed will be less than its insertion time, i.e., at most $O(f(N)P_S(N)/N)$. But sometimes we need to rebuild the complete structure after a deletion. This takes at most $P_S(k)$ were k is the number of points left in the structure (using that P_S is at least linear). Divided over the number of deletions since the last rebuilding, which is k too, this makes for an extra charge of $\frac{P_S(k)}{k} \leq \frac{P_S(N)}{N}$ per deletion. Note that we no longer "count" in a monotone manner in the primary structure, but this can be shown not to affect the time-estimate on insertions by more than a constant factor (cf. [5]).

□

Another class of decomposable searching problems for which one can process deletions efficiently was defined by Overmars and van Leeuwen [3], later redefined in a slightly more general way by van Leeuwen and Maurer [5].

Definition 4.2. A decomposable searching problem, together with a static data-structure S for it, is called a DD-searching problem if and only if S allows one to delete arbitrary objects or to replaced them by some kind of dummy object that will not affect later queries, without causing an increase in the future deletion or querytime. Let $D_S(n)$ be the time needed for such a "deletion" in a structure of n points.

Note that the querytime may remain unchanged, despite the fact that more and more points get deleted from S (the structure will go "out of balance"). (Van Leeuwen and Maurer [5] call these problems RD-decomposable.)

Theorem 4.2. Given a DD-searching problem and a dynamization method of the sort described above, one can perform deletions in $O(D_S(N) + P_S(N)/N)$ average time, without changing the bounds on the query and insertion time.

Proof.

We will only give a brief sketch of the method suggested in [5]. When we want to delete a point we first locate the block it is in and delete it (or replace it by a dummy). We ignore the time for any dictionary look-up needed. The deletion of the point from its block takes at most $D_S(N)$. When the proportion of dummies in the whole structure has become too big as a result of the last deletion, then we rebuild the complete structure. This takes at most $P_S(N)$, which makes $O(P_S(N)/N)$ per deletion. Because the number of dummies is kept less than a constant fraction of the total number of points the average insertion and querytimes can be shown to remain essentially unaltered.

□

One can show that the deletion time is in fact $O(D_S(m) + P_S(m)/m)$ where m is the maximal number of points in the set at any time. Some interesting examples of DD-searching problems can be found in [3] and [5].

Overmars and van Leeuwen [3] also define a subclass of decomposable searching problems, called MD-searching problems, for which a merging technique can be used to obtain even faster insertions. This technique can also be used for our second class of dynamization methods to save a factor of log N in insertion time for MD-searching problems.

6. References.

[1] Bentley, J.L., Decomposable searching problems, Inf. Proc. Lett. 8 (1979) pp. 244-251.

[2] Knuth, D.E., The art of computer programming, vol. 2: Semi-numerical algorithms, Addison-Wesley, Reading, Mass., 1969.

[3] Overmars, M.H. and J. van Leeuwen, Two general methods for dynamizing decomposable searching problems, Techn. Rep. RUU-CS-79-10, Dept. of Computer Science, University of Utrecht, the Netherlands, 1979.

[4] Saxe, J.B. and J.L. Bentley, Transforming static data-structures into dynamic structures, Proc. 20th Annual IEEE symp. on Foundations of Computer Science, 1979, pp. 148-168.

[5] Van Leeuwen, J. and H.A. Maurer, Dynamic systems of static data-structures, Bericht 42, Inst. f. Informationsverarbeitung, TU Graz, Austria, 1980.