WORST-CASE OPTIMAL INSERTION AND DELETION METHODS FOR DECOMPOSABLE SEARCHING PROBLEMS

Mark H. Overmars and Jan van Leeuwen

RUU-CS-80-10
November 1980

Rijksuniversiteit Utrecht

Vakgroep informatica

Princetonplein 5
Postbus 80.002
3508 TA Utrecht
Telefoon 030–53 1454
The Netherlands

81 3/11

WORST-CASE OPTIMAL INSERTION AND DELETION METHODS FOR DECOMPOSABLE SEARCHING PROBLEMS

Mark H. Overmars and Jan van Leeuwen

Technical Report RUU-CS-80-10

November 1980

Department of Computer Science
University of Utrecht
P.O. Box 80.002, 3508 TA Utrecht
the Netherlands

WORST-CASE OPTIMAL INSERTION AND DELETION METHODS FOR DECOMPOSABLE SEARCHING PROBLEMS

Mark H. Overmars* and Jan van Leeuwen

Department of Computer Science, University of Utrecht
P.O.Box 80.002, 3508 TA Utrecht, the Netherlands

Abstract. Two years ago Saxe and Bentley [11] proposed a number of ways to process insertions in dynamic structures of static data structures for decomposable searching problems. The question to handle both insertions and deletions efficiently has been open since, although solutions were obtained for restricted classes. In this paper we settle the original question and develop a number of techniques which, when super-imposed on the two most prominent dynamizations known to date, yield worst-case optimal insertion and deletion methods for general decomposable searching problems.

Keywords and phrases. Decomposable searching problems, dynamization, insertion, deletion, worst-case bound.

1. Introduction.

The design of efficient algorithms and data structures often aims at providing solutions to object-oriented searching problems. Let $Q(V)$ denote the answer to a searching problem Q over a set V.

Definition. Q is called decomposable if and only if for each partition $V_1 \cup V_2$ of V one has $Q(V) = \square\,(Q(V_1), Q(V_2))$ for an efficiently computable operator $\square$.

Decomposable searching problems were first introduced by Bentley [1], in his study of general searching problems for multi-dimensional point sets. While many data structures used in this field are highly inflexible and practical only for static sets of points, Bentley noted that for decomposable searching problems one can avoid storing all points in one massive data structure and maintain the set as a "dynamic" system of disjoint blocks in-

*: The work of this author was supported by the Netherlands Organization for the Advancement of Pure Research (Z.W.O.)

stead (where each block is statically structured). Saxe and Bentley [11] gave a number of methods to perform insertions in such systems, achieving different levels of efficiency. Soon after, van Leeuwen and Wood [14] (see also [4]) obtained a method that supported both insertions and deletions. Still, the question to handle insertions and deletions together in a "best possible" way has remained open since.

Several authors have addressed the problem for restricted classes, usually of searching problems which had data structures that permitted "quick" deletions, without substantially worsening the search characteristics of the structure (see e.g. Overmars and van Leeuwen [9]). The most effective techniques here were developed by van Leeuwen and Maurer [13] (who obtained good average bounds) and by Overmars and van Leeuwen [10] (who succeeded in proving that these bounds are valid even as worst-case bounds). From the other end Mehlhorn [5] (extending results of [11]) addressed questions of optimality and Mehlhorn and Overmars [6] came up with a fairly direct scheme to achieve the proven lowerbounds when only insertions are considered. In this paper we exploit all insights gained and present the necessary modifications and additional techniques required to finally solve the original problem. Many dynamizations (including dynamic solutions to range querying and nearest neighbor searching) that were obtained by especially designed methods before, will follow immediately from our main result here.

The common paradigm in many existing data structures for decomposable searching problems is that neither insertions nor deletions actually need to restore the "balance" of a structure immediately when they occur, as long as the structure remains "in reasonable shape".

Definition. Deletions and/or insertions are called "weak" if the routines to carry them out on an admissible structure of n points merely guarantee that after αn deletions ($\alpha < 1$) and/or βn insertions the query time on the resulting set of m elements is bounded by $k_{\alpha\beta}Q(m)$, for some constant $k_{\alpha\beta}$ depending solely on α and β and $Q(m)$ the querytime on a structure of m points on which no weak updates are performed.

In the sequel, (weak)deletions will be of special concern. We need the following notations, given a structure S of n points:

$Q_S(n)$ = the time needed to solve Q on S,
$P_S(n)$ = the time needed to build S,
$D_S(n)$ = the time needed to perform a deletion on S,

$I_S(n)$ = the time needed to perform an insertion on S,
$WD_S(n)$ = the time needed to perform a weak deletion on S,
$WI_S(n)$ = the time needed to perform a weak insertion on S.

All bounds are worst-case bounds. We assume that all functions are non-decreasing and smooth and that P_S is at least linear.

2. An optimal dynamization method.

One of the main tools we will use is the following general theorem that shows how to turn weak updates into clean updates, i.e., updates that maintain structures in a "balanced" form, without much loss of time.

Theorem 1. Given a structure S for a searching problem Q, there is a structure S_1 for Q such that

$$Q_{S_1}(n) = O(Q_S(n))$$

$$D_{S_1}(n) = O(WD_S(n) + P_S(n)/n)$$

$$I_{S_1}(n) = O(WI_S(n) + P_S(n)/n)$$

Proof

S_1 will consist of at most 2 S-structures: OLD-MAIN and MAIN. Normally only MAIN exists. When the number of transactions on MAIN exceeds half its initial size, MAIN is made into OLD-MAIN and a construction is initiated to build a new (i.e., rebalanced) MAIN from it. Meanwhile insertions, deletions and queries continue on OLD-MAIN, until the new MAIN under construction can take over. Assuming there were n_0 points when the construction began, we shall see to it that MAIN can take over after at most $\frac{1}{4}n_0$ transactions (weak deletions and insertions) have occured, which will guarantee that no next reconstruction is needed before it is finished.

The idea is to do $P_S(n_0)/\frac{1}{4}n_0$ work on the construction during each of the next $\frac{1}{4}n_0$ transactions. For the new MAIN to be valid, though, the insertions and deletions carried out on OLD-MAIN during its construction must be performed on it as well. To this end we buffer these transactions in BUF and speed up the construction by performing an additional amount of $WD_S(n_0 + \frac{1}{4}n_0)$ work with each deletion and $WI_S(n_0 + \frac{1}{4}n_0)$ work with each insertion (considering that the size of MAIN becomes at most $n_0 + \frac{1}{4}n_0$ while performing buffered updates). This ensures that when the construction of MAIN is finished, enough time is left to perform all buffered transactions together with the next incoming transactions. Until a total of $\frac{1}{4}n_0$ transactions is reached. While the transactions continue to be performed on OLD-MAIN (so it remains in use for query answering), they line up in BUF as well and $WD_S(n_0 + \frac{1}{4}n_0) + P_S(n_0)/\frac{1}{4}n_0$ or $WI_S(n_0 + \frac{1}{4}n_0) + P_S(n_0)/\frac{1}{4}n_0$ work is spent on whatever transactions occur up front in BUF. A simple argument shows that BUF will be empty (so MAIN can take over) within $\frac{1}{4}n_0$ transactions, as desired.

Observe that on no block that begins as MAIN and eventually becomes OLD-MAIN before it is discarded, more than $\frac{3}{4}n_0$ weak deletions or insertions are performed during its life-time. Hence $Q_{S_1}(n) = O(Q_S(n))$.

□

It easily follows from the proof that the $O(P_S(n)/n)$ work need not be paid for deletions when they are clean (i.e., when $WD_S = D_S$) nor for insertions when they are clean (i.e., when $WI_S = I_S$). Theorem 1 leads to many direct results in obtaining efficient dynamic data structures for searching problems (see section 3 example a, and Overmars [8]) but in this paper we are mainly concerned with its application to the dynamization of decomposable searching problems.

There have essentially been two successful methods to dynamize decomposable searching problems until now.

METHOD A: the logarithmic method

The set is partitioned as a collection of $\leq f(n)$ blocks of exponentially increasing sizes, with each block built as a static structure. Insertions are always performed at the "low" end, normally at the expense of rebuilding a few small blocks to properly include the new points. Deletions are directly performed in the blocks points belong to (and thus often are weak deletions), with only occasionally a need to clean up the whole structure. (See Bentley [1], Saxe and Bentley [11], Overmars and van Leeuwen [9], van Leeuwen and Maurer [13], Mehlhorn and Overmars [6].)

METHOD B: the equal blocks method

The set is partitioned as a collection of $\leq g(n)$ blocks of about equal sizes, with each block built as a static structure. Insertions are always performed on the smallest block, deletions on the block a point belongs to. (Thus insertions and deletions will often be "weak".) Only occasionally the structure is reconfigured, to adjust the number of blocks and their sizes. (See Maurer and Ottman [4], van Leeuwen and Wood [14], van Leeuwen and Maurer [13].)

The following two theorems show the efficiency of the two methods.

Theorem A. Given a structure S for a decomposable searching problem Q, it can be dynamized to a structure S_1 for Q with

$$Q_{S_1}(n) \le O(f(n))\ Q_S(n)$$

$$D_{S_1}(n) \le O(WD_S(n) + \log n + P_S(n)/n)$$

$$I_{S_1}(n) \le \begin{cases} O(\log n/\log \frac{f(n)}{\log n})\ P_S(n)/n & \text{if } f(n) = \Omega(\log n) \\ O(f(n).n^{\frac{1}{f(n)}})\ P_S(n)/n & \text{if } f(n) = O(\log n) \end{cases}$$

Proof

The bounds on query and insertion time (the latter only as an average) were given in Mehlhorn and Overmars [6] who constructed the appropriate structure S_1 for it. As S_1 consists of a number of S structures one can perform weak deletions on S_1 in the following way. First find the block to which the point for deletion belongs. If we keep track of the block-name for each point of the set in a dictionary, then this requires only O(log n) time. Secondly, perform a weak deletion with the point on that block, which takes $WD_S(n)$. It follows that $WD_{S_1}(n) = O(WD_S(n) + \log n)$ and we can apply theorem 1 to obtain $D_{S_1}(n) = O(WD_S(n) + \log n + P_S(n)/n)$. (One easily shows that $P_{S_1}(n) \le P_S(n)$.) Turning the average bounds for insertions into worst-case bounds can be done in a way very similar to the method described in Overmars and van Leeuwen [10].

□

Theorem B. Given a structure S for a decomposable searching problem Q, one can dynamize it to a structure S_1 for Q with

$$Q_{S_1}(n) \le O(g(n))\ Q_S(n/g(n))$$

$$D_{S_1}(n) \le O(WD_S(n/g(n)) + \log n + P_S(n)/n)$$

$$I_{S_1}(n) \le O(WI_S(n/g(n)) + \log n + P_S(n)/n)$$

Proof

S_1 consists of g(n) blocks of size n/g(n). To perform a weak deletion on S_1 we locate the block the point is in (by using a dictionary) and perform a weak deletion at that block. This clearly takes $O(WD_S(n/g(n)) + \log n)$. To perform a weak insertion on S_1 we perform a weak insertion on the smallest block and update the dictionary. (One easily shows that these operations are indeed weak updates.) The bounds as stated now follow by applying theorem 1.

□

Since theorems A and B use an "arbitrary" structure S with known characteristics as a start, we can apply e.g. theorem B to the structure S_1 resulting from theorem A. It gives a structure S_2 with the following characteristics:

$$Q_{S_2}(n) \leq O(f(n).g(n))\ Q_S(n/g(n))$$

$$D_{S_2}(n) \leq O(WD_S(n/g(n)) + \log n + P_S(n)/n)$$

$$I_{S_2}(n) \leq \begin{cases} O(\log n/\log \frac{f(n)}{\log n})\ P_S(n)/n & \text{if } f(n) = \Omega(\log n) \\ O(f(n).n^{\frac{1}{f(n)}})\ P_S(n)/n & \text{if } f(n) = O(\log n) \end{cases}$$

Thus S_2 has the best insertion and deletion times of both methods, but the penalty factor in query time has become $f(n).g(n)$. A different technique of "marrying" A and B will reduce this to $f(n) + g(n)$, which is best possible. We give a very general result first.

Theorem 2. Given a (dynamic) structure S_1 composed of $O(f(n))$ S-blocks structured for a decomposable searching problem Q, there is a structure S_3 with

$$Q_{S_3}(n) \leq O(f(n) + g(n))\ Q_S(n/g(n))$$

$$D_{S_3}(n) \leq O(WD_S(n/g(n)) + \log n + P_S(n)/n)$$

$$I_{S_3}(n) \leq I_{S_1}(n) + O(P_S(n)/n)$$

Proof

Split each block of S_1 of size $> n/g(n)$ into sub-blocks (again structured "like S") of size $n/g(n)$. This leads to at most $f(n) + g(n)$ (sub)-blocks of size $\leq n/g(n)$. The bound on the query time follows. Weak deletions are carried out per block, insertions are as they were for S_1 except that new blocks are immediately split in sub-blocks of size $n/g(n)$. The resulting structure S_2 has

$$Q_{S_2}(n) \leq O(f(n) + g(n))\ Q_S(n/g(n))$$

$$WD_{S_2}(n) \leq WD_S(n/g(n)) + O(\log n)$$

$$WI_{S_2}(n) \leq O(I_{S_1}(n))$$

Applying theorem 1 to S_2 gives a structure S_3 of the desired characteristics. □

Using for S_1 the specific structure resulting from theorem A, we immediately obtain the following powerful result.

Theorem C. Given a structure S for a decomposable searching problem Q, one can dynamize it to a structure S' with

$$Q_{S'}(n) \leq O(f(n) + g(n))\ Q_S(n/g(n))$$

$$D_{S'}(n) \leq O(WD_S(n/g(n)) + \log n + P_S(n)/n)$$

$$I_{S'}(n) \leq \begin{cases} O(\log n/\log \frac{f(n)}{\log n}).P_S(n)/n & \text{if } f(n) = \Omega(\log n) \\ O(f(n).n^{\frac{1}{f(n)}}).P_S(n)/n & \text{if } f(n) = O(\log n) \end{cases}$$

It combines the best results and optimal worst-case bounds (cf. Mehlhorn [5]) in one single method.

3. Some applications.

Using proper choices for f and g, theorem C can be tuned in many different ways. The result is useful to obtain dynamic solutions to many (static) searching problems which hitherto were not dynamized properly or only so by apparent ad hoc techniques.

a. Range querying

Willard [15] (see also Lueker [3]) was first to obtain an intricate and fully dynamic structure for the d-dimensional range query problem. The data structure he devised could be built in $P_S(n) = O(n \log^{d-1} n)$ and achieved $Q_S(n) = O(\log^d n)$, ignoring the time to actually output the data, $D_S(n) = O(\log^d n)$ and $I_S(n) = O(\log^d n)$. It can be shown (see van Leeuwen and Maurer [13]) that the structure allows weak deletions in time $WD_S(n) = O(\log^{d-1} n)$, by merely removing all links to an element everywhere it occurs. Applying theorem 1 yields a dynamic solution to range searching with

$$Q_{S_1}(n) = O(\log^d n)$$

$$D_{S_1}(n) = O(\log^{d-1} n)$$

$$I_{S_1}(n) = O(\log^d n)$$

This result is better than any previous bounds known. Theorem 1 has a number of intriguing further applications (see Overmars [8]).

b. Nearest neighbor searching in the plane (compare van Leeuwen and Maurer [13]).

Known static solutions use structures with $P_S(n) = O(n \log n)$ and $Q_S(n) = O(\log n)$ (Shamos [12], Kirkpatrick [2]). Van Leeuwen and Maurer [13] used it to obtain a dynamic solution with

$$Q_S(n) = O(\sqrt{n.\log n})$$

$$D_S(n) = O(\sqrt{n.\log n})$$

$$I_S(n) = O(\sqrt{n})$$

, where the time bounds for deletions and insertions were obtained as averages.

Overmars [7] recently obtained a dynamic structure for nearest neighbor searching that achieves the following bounds:

$$Q_S(n) = O(\log n)$$

$$D_S(n) = O(n)$$

$$I_S(n) = O(n)$$

Using $f(n) = g(n) + \sqrt{n}/\sqrt{\log n}$, theorem C yields a remarkable dynamic structure for nearest neighbor searching that achieves the following worst-case bounds

$$Q_S(n) = O(\sqrt{n \log n})$$

$$D_S(n) = O(\sqrt{n \log n})$$

$$I_S(n) = O(\log n)$$

Different choices of f and g can lower the query time at the expense of a higher deletion (and insertion) time.

4. References.

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

[2] Kirkpatrick, D.G., Optimal search in planar subdivisions, preprint, Dept. of Computer Science, UBC, Vancouver, Canada, 1979.

[3] Lueker, G.S., A transformation for adding range restriction capability to dynamic data structures for decomposable searching problems, Techn. Rep., Dept. of Inform. and Computer Science, University of California, Irvine, 1978.

[4] Maurer, H.A. and Th. Ottman, Dynamic solutions of decomposable searching problems, Bericht 33, Inst. f. Informationsverarb. TU Graz, 1979.

[5] Mehlhorn, K., Lowerbounds on the efficiency of static to dynamic transformations of data structures, Techn. Rep. A 80/05, Fachber. 10, Universität des Saarlandes, Saarbrücken, 1980.

[6] Mehlhorn, K. and M.H. Overmars, Optimal dynamization of decomposable searching problems, Techn. Rep. A 80/.., Fachber. 10, Universität des Saarlandes, Saarbrücken, 1980.

[7] Overmars, M.H., Dynamization of order decomposable set problems, Techn. Rep. RUU-CS-80-9, University of Utrecht, Utrecht, 1980.

[8] Overmars, M.H., Transforming semi-dynamic data structures into dynamic structures, 1981 (in preparation).

[9] Overmars, M.H. and J. van Leeuwen, Two general methods for dynamizing decomposable searching problems, Techn. Rep. RUU-CS-79-10, University of Utrecht, Utrecht, 1979 (to appear in Computing).

[10] Overmars, M.H. and J. van Leeuwen, Dynamizations of decomposable searching problems yielding good worst case bounds, Proc. of the 5 GI Fachtagung Theoretische Informatik, Springer Verlag, 1981 (to appear).

[11] Saxe, J.B. and J.L. Bentley, Transforming static data structures to dynamic structures, Proc. 20th Ann. IEEE Symp. on Foundations of Computer Science, 1979, pp. 148-168.

[12] Shamos, M.I., Computational geometry, Ph. D. Thesis, Yale University, 1978 (to be published).

[13] van Leeuwen, J. and H.A. Maurer, Dynamic systems of static data structures, Bericht 42, Inst. f. Informationsverarb., TU Graz, 1980.

[14] van Leeuwen, J. and D. Wood, Dynamization of decomposable searching problems, Inf. Proc. Lett. 10 (1980) 51-56.

[15] Willard, D.E., The super B-tree algorithm, TR-03-79, Aiken Comput. Lab., Harvard University, Cambridge, USA, 1979.