On Dynamic k–d Half–Space Range Reporting[1]

Bradford G. Nickerson[2] Yunlan Pan[3]

Abstract

We consider the problem of reporting all points in a positive half–space h^* (bounded by a query hyperplane h) from a set S of n points in k–dimensional (k–d) space E^k; i.e. find and report $S \cap h^*$. Previous dynamic solutions (supporting insertion and deletion of points) are simplified, and the previously best known dynamic update time is improved. For $t = |S \cap h^*|$, we show that with $O(k\,n \log n)$ preprocessing time and $\Theta(k\,n)$ space, the k–d half–space range reporting problem can be solved in $O(k\,n)$ time with dynamic update time of $O(k \log n)$. Experimental results with k = 5, 7 and 10 for randomly generated data points with n = 10,000, 50,000 and 100,000 show range reporting times between 0.001 and 0.20 seconds, with up to 90% of the points in range.

Keywords: Algorithms, data structures, range search, computational geometry

Introduction

As pointed out in [1], half–space range reporting is important for a number of applications, including ray shooting, nearest/farthest neighbour, dynamic linear optimization queries, and others. Agarwal and Matousek's [1] solution to this problem takes $O((n/m^{\lfloor k/2 \rfloor})\log n + t)$ query time and $O(m^{(1+\varepsilon)}/n)$ amortized update time ($k \geq 3$), for a given parameter m, $n \leq m \leq n^{\lfloor k/2 \rfloor}$ and an arbitrarily small constant ε. Their solution builds on previous work [5, 6] and contains the interesting result that as k increases, the upper bound on query time decreases. This is in contrast to the known k–d orthogonal range reporting time of $\Omega(k \log n + t)$ for the tree–based model of computation [2].

Agarwal and Matousek's algorithm is complex, requiring the partitioning of the points S into q subsets S_i for $q = \lceil \log_\beta n \rceil$ such that S_i contains at most $(\beta - 1)\beta^{\,i}$ points. In addition, their algorithm is carried out in a dual setting, where the points in S are represented as a set H of n hyperplanes, and the half–space range reporting problem is answered by reporting the hyperplanes of H lying above a query point.

Our approach is simpler, and the relatively straight–forward data structure used for storing the points results in the faster dynamic update time of $O(k \log n)$. For our algorithm, the k–d half–space range reporting time is $O(k\,n)$.

Data Structure

The data structure used here is based on the 1–3 deterministic skip list (DSL) of [7] and the 2–d search skip list of [8]. We call the data structure the "half–space k–d search skip list". The node structure for a single node in the k–d case is shown in Figure 1. Figure 2 shows an example half–space 2–d search skip list.

[1] This research was supported in part by the Natural Sciences and Engineering Research Council of Canada and the UNB Faculty of Computer Science.
[2] corresponding author: Faculty of Computer Science, University of New Brunswick, P.O. Box 4400, Fredericton, N.B., Canada, E3B 5A3, E–mail: bgn@unb.ca
[3] 9051 E. Shorewood Dr., Mercer Island, WA 98040, USA

datapoint (x_1, x_2, ... , x_k)	
min. x_1 value in left subtree	max. x_1 value in left subtree
min. x_2 value in left subtree	max. x_2 value in left subtree
• • •	• • •
min. x_k value in left subtree	max. x_k value in left subtree
down pointer	right pointer

```
type
   NodePtr = ^Node;
   Node = record
      coord : array[3,k] of coordinate;
      down, right : NodePtr;
   end;
```

Figure 1. Half–space *k*–d search skip list node structure, and a Pascal record structure corresponding to it.

algorithms to mark the beginning and end of data. Each leaf node contains one datapoint; these are maintained in order of x_1 values. Interior nodes have a datapoint equal to that of the rightmost leaf level datapoint in the left subtree of this interior node.

Theorem 1. The space S(*n*, *k*) required to store a set of *n* points in a half–space *k*–d search skip list is $\Theta(k\,n)$.

Proof: Follows from the fact that each node has $(3k + 2)$ data values, and the fact that the maximum height of the half–space *k*–d search skip list is $\lfloor \log_2 n \rfloor + 2$.

Theorem 2. The time I(*n*, *k*) required to insert a new point into a half–space *k*–d search skip list is $O(k \log n)$.

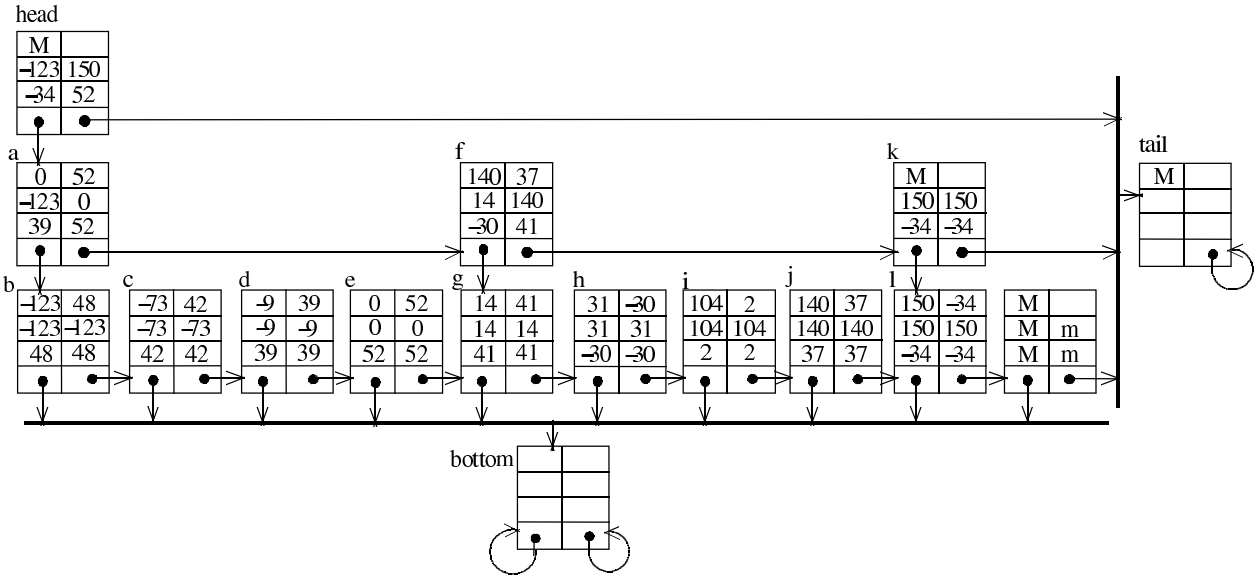

Figure 2. Example half–space *2*–d search skip list.

Note that M stands for the maximum coordinate value and m stands for the minimum coordinate value. The down and right pointers point to the left and right subtrees, respectively. The algorithms for insertion of points into and deletion of points from the half–space *k*–d search skip list are not given here as they are similar to the DSL and search skip list algorithms. Special nodes head, tail and bottom are used in these

Theorem 3. The preprocessing time P(*n*, *k*) required to construct a half–space *k*–d search skip list of *n* points is $O(k\,n \log n)$.

Theorem 4. The time D(n, k) required to delete a point from a half–space k–d search skip list is O($k \log n$).

The proofs of theorems 2, 3 and 4 are omitted due to similarities with the DSL and the *2*–d search skip list.

2–d Half–Space Range Reporting

In E^2, a hyperplane h is defined by the equation $ax + by - c = 0$, for $a \neq 0$. S is a subset of E^2. For $b \neq 0$, let $y' = (-ax_{min} + c)/b$, and $y'' = (-ax_{max} + c)/b$ (see Figure 3). Similarly, let $x' = (-by_{min} + c)/a$, and $x'' = (-by_{max} + c)/a$.

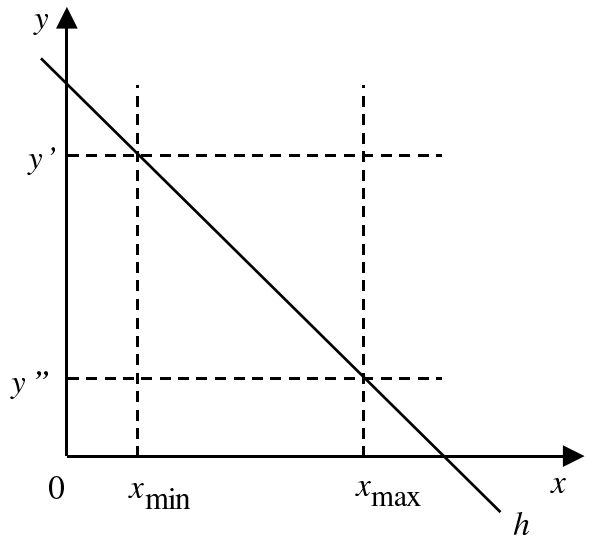

Figure 3. *2*–d half space range reporting.

Lemma 1. For any point $(x, y) \in S$,
(i) if $b > 0$, then
$y < \min(y', y'')$ implies $ax + by - c < 0$,
and
$y > \max(y', y'')$ implies $ax + by - c > 0$.
(ii) if $b < 0$, then
$y < \min(y', y'')$ implies $ax + by - c > 0$,
and
$y > \max(y', y'')$ implies $ax + by - c < 0$.

Proof: For any point $(x, y) \in S$, let
$y_0 = (-ax + c)/b$, $y_1 = \min(y', y'')$,
$y_2 = \max(y', y'')$. As illustrated in Figure 3, we have $y_1 \leq y_0 \leq y_2$.
(i) For $b > 0$, if $y < y_1$, then
$y < y_1 \leq y_0 = (-ax + c)/b$, or $by < -ax + c$
giving $ax + by - c < 0$;
if $y > y_2$, then
$y > y_2 \geq y_0 = (-ax + c)/b$, or $by > -ax + c$
giving $ax + by - c > 0$.
(ii) For $b < 0$, if $y < y_1$, then
$y < y_1 \leq y_0 = (-ax + c)/b$, or $by > -ax + c$
giving $ax + by - c > 0$;
if $y > y_2$, then
$y > y_2 \geq y_0 = (-ax + c)/b$, or $by < -ax + c$
giving $ax + by - c < 0$.

Lemma 2. For any point $(x, y) \in S$,
(i) if $a > 0$, then
$x < \min(x', x'')$ implies $ax + by - c < 0$,
and
$x > \max(x', x'')$ implies $ax + by - c > 0$.
(ii) if $a < 0$, then
$x < \min(x', x'')$ implies $ax + by - c > 0$,
and
$x > \max(x', x'')$ implies $ax + by - c < 0$.

Proof: Similar to Lemma 1.

The algorithm for *2*–d half–space range reporting can now be given as shown in Figure 4.

Note that for the algorithm in Figure 4, x_{min}, x_{max}, y_{min}, y_{max} are stored in R^.coord[2,1],

```
1   procedure halfSpReport( h : hyperplane; R : NodePtr; max1 : coordinate)
2   begin
3       if (R^.coord[1,1] = M and R points to the bottom node) then return;
4       else if (R^.coord[1,1] = M and R does not point to the bottom and R ≠ tail node) then
5           halfSpReport(h, R^.down, R^.coord[1,1])
6       else if (R is not the tail or bottom node) then
7           begin
8               if (R points to the bottom node) then  {we are at the leaf level}
9                   begin
10                      if this point is in range, report it;
11                      if (R^.coord[1,1] < max1 ) then halfSpReport(h, R^.right, R^.coord[1,1])
12                  end;
13              else  { we are not at the leaf level }
14                  begin
15                      CheckNode(h, xmin, xmax, ymin, ymax, in, prune);
16                      if (in) then report this subtree as being in range
17                      else if (not prune) then halfSpReport(h, R^.down, R^.coord[1,1]);
18                      if (R^.coord[1,1] < max1 ) then halfSpReport(h, R^.right, R^.coord[1,1])
19                  end;
20          end;
21  end; { halfSpReport }
```

Figure 4. Pascal pseudo–code algorithm for *2*–d half–space range reporting.

R^.coord[3,1], R^.coord[2,2], and R^.coord[3,2], respectively. The argument max1 contains the maximum x_1 value in the left subtree (on the initial call it is set to M).

```
procedure CheckNode( h : hyperplane; xmin, xmax, ymin, ymax : coordinate; var in, prune : boolean)
begin
    calculate x', x'';
    in := prune := false;
    if (a > 0 and xmax < min(x', x'')) or (a < 0 and xmin > max(x', x'')) then prune := true
    else if (a > 0 and xmin > max(x', x'')) or (a < 0 and xmax < min(x', x'')) then in := true
    else {check b}
        if (b ≠ 0) then
            begin
                calculate y', y'';
                if (b > 0 and ymax < min(y', y'')) or (b < 0 and ymin > max(y', y''))
                    then prune := true
                else if (b > 0 and ymin > max(y', y'')) or (b < 0 and ymax < min(y', y''))
                    then in := true
            end;
end; {CheckNode}
```

Figure 5. CheckNode algorithm to determine if the left subtree can be completely pruned or is completely inside the positive half–space.

As an illustration of how the algorithm works, consider the nine points and *2*–d half–space of Figure 2. Figure 6 shows the nine points plotted in E^2, along with a hyperplane
$x - y + 41 = 0$.

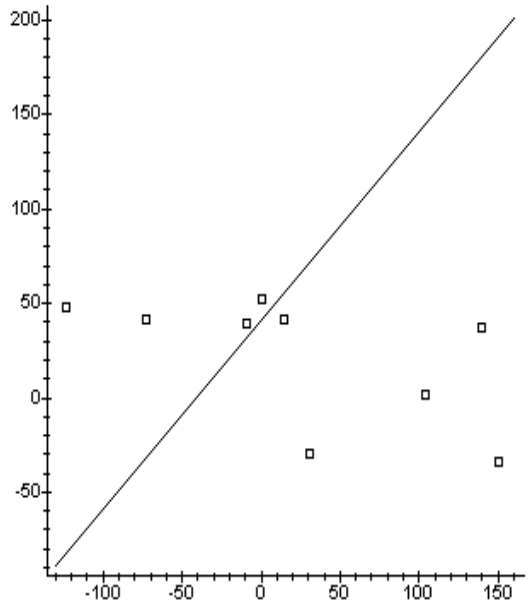

Figure 6. Plot of the *2*–d points of Figure 2 and the hyperplane $x - y + 41 = 0$.

The algorithm starts by considering the head node. Computing (x', x'') and (y', y'') does not allow us to report or prune the entire skip list. Checking the left subtree rooted at node "a" again does not report or prune the entire left subtree. Checking the leaves b, c, d, and e determines that none of them are in range and they are not reported. Checking the right sibling (node "f") determines that $a > 0$ and $x_{min} > \max(x', x'')$, causing all points in the left subtree (i.e. those in nodes g, h, i, and j) to be reported in range. A similar finding at node "k" causes the point in node l to be reported in range.

k–d Half–Space Range Reporting

In E^k, a hyperplane h is defined by the equation $AX^T = c$, where $A = (a_1, a_2, \ldots, a_k)$ represents the hyperplane coefficients, $X = (x_1, x_2, \ldots, x_k)$ represents the k coordinates, and c is a constant. Given a set of points S in E^k, we want to find and report the points of S in the positive half–space h^* of h. We assume that no points in S lie on the hyperplane.

Definition 1. We say $p = (x_1, x_2, \ldots, x_k)$ belongs to positive half–space h^*, denoted $p \in h^*$, if $A\,(x_1, x_2, \ldots, x_k)^T - c > 0$. The negative half–space $h^{\#}$ of h can be defined by reversing the inequality.

Definition 2. For a given set of points S in E^k, and any integer i, $1 \le i \le k$, $x_i^{\min}$ is defined as the minimum value of the x_i coordinates of all the points in S. Similarly, $x_i^{\max}$ is defined as the maximum value of the x_i coordinates of all the points in S.

Definition 3. For a given set of points S in E^k, we define two k–element arrays mins and maxs. For any integer i, $1 \le i \le k$,

$$\text{mins}[i] = \begin{cases} -a_i x_i^{\min}, & \text{if } a_i < 0 \\ 0, & \text{if } a_i = 0 \\ -a_i x_i^{\max}, & \text{if } a_i > 0 \end{cases}$$

$$\text{maxs}[i] = \begin{cases} -a_i x_i^{\max}, & \text{if } a_i < 0 \\ 0, & \text{if } a_i = 0 \\ -a_i x_i^{\min}, & \text{if } a_i > 0 \end{cases}$$

Definition 4. For a given set of points S in E^k, we define two discrete functions f_{mins} and

f_{maxs} on the set {1, 2, ..., k}. For any integer i, $1 \le i \le k$,

$$f_{mins}(i) = \sum_{j \neq i} \min s[j], \text{ and}$$
$$f_{maxs}(i) = \sum_{j \neq i} \max s[j].$$

Lemma 3. Given a set of points S in E^k, then for any point $p = (x_1, x_2, \ldots, x_k) \in S$, we have $mins[i] \le -a_i x_i \le maxs[i]$, $\forall\, i \in \{1, 2, ..., k\}$.
Therefore

$$f_{mins}(i) \le f_{maxs}(i) \;\forall\, i \in \{1, 2, ..., k\}.$$

Proof: From definition 4 and the following facts:

For $1 \le i \le k$, $x_i^{min} \le x_i \le x_i^{max}$.

If $a_i < 0$, we have

$-a_i x_i^{min} \le -a_i x_i \le -a_i x_i^{max}$; i.e.

$mins[i] \le -a_i x_i \le maxs[i]$.

Similarly, if $a_i \ge 0$, we have

$-a_i x_i^{min} \ge -a_i x_i \ge -a_i x_i^{max}$; i.e.

$mins[i] \ge -a_i x_i \ge maxs[i]$. QED

Now, let a hyperplane h in E^k be defined by $AX^T = c$, S be a subset of E^k, and i be an integer such that $1 \le i \le k$. If $a_i \neq 0$, let

$$x_i' = (f_{mins}(i) + c)/a_i, \text{ and}$$
$$x_i'' = (f_{maxs}(i) + c)/a_i.$$

Lemma 4. For any point
$p = (x_1, x_2, \ldots, x_k) \in S$,

(i) if $a_i > 0$, then $x_i < x_i'$ implies $p \in h^{\#}$, and
$x_i > x_i''$ implies $p \in h^*$.

(ii) if $a_i < 0$, then $x_i < x_i'$ implies $p \in h^*$,
and $x_i > x_i''$ implies $p \in h^{\#}$.

Proof: If $a_i \neq 0$, and i an integer such that $1 \le i \le k$, then, by Lemma 3, we have $x_i' \le x_i''$. For any point
$p = (x_1, x_2, \ldots, x_k) \in S$, let
$x_i^0 = (\sum_{j \neq i} (-a_j x_j) + c) / a_i$. Again, by Lemma 3 we have $x_i' \le x_i^0 \le x_i''$.

(i) For $a_i > 0$, if $x_i < x_i'$ then

$x_i < x_i' \le x_i^0 = (\sum_{j \neq i} (-a_j x_j) + c) / a_i$, or

$a_i x_i < \sum_{j \neq i} (-a_j x_j) + c$, or

$A(x_1, x_2, \ldots, x_k)^T - c < 0$. Thus, $p \in h^{\#}$.

For $a_i > 0$, if $x_i > x_i''$ then

$x_i > x_i'' \ge x_i^0 = (\sum_{j \neq i} (-a_j x_j) + c) / a_i$, or

$a_i x_i > \sum_{j \neq i} (-a_j x_j) + c$, or

$A(x_1, x_2, \ldots, x_k)^T - c > 0$. Thus, $p \in h^*$.

(ii) For $a_i < 0$, similar arguments hold. QED

From Lemma 4, the algorithm for k–d half–space range reporting can now be defined. It is identical to the algorithm of Figure 4, except for changes to CheckNode. The arguments x_{min}, x_{max}, y_{min}, y_{max} are replaced by two arrays of size k to hold the values of x_i' and x_i''. CheckNode also uses a loop from 1 to k to check if a left subtree is completely in range or completely pruned.

Analysis

Theorem 5. Given a set of n points in E^k, the worst case time for k–d half–space range reporting using the k–d half–space search skip list is $O(k\,n)$.

Proof: The worst case for k–d half–space range reporting is illustrated in Figure 7. In this case, there are no points in range, but the k–d half–space range reporting algorithm must search n nodes to make this determination. The points are organized in such a way that $x_1^{\min} \le x_1 \le x_1^{\max}$, and $x'_j < x_j < x''_j$, for $j = 2, 3, \ldots, k$. Thus, the algorithm never prunes as x_j is always in the range (x'_j, x''_j), until the leaf level is reached. This means that $O(n)$ nodes are checked to determine that no points are in the desired half space, and each node requires $O(k)$ operations. QED

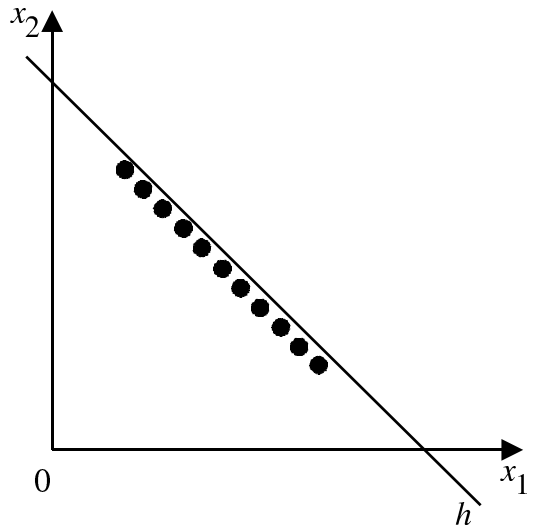

Figure 7. Illustration of worst case 2–d half–space range reporting with the 2–d half–space search skip list.

Experimental Results

Experiments were performed for values of k = 5, 7 and 10, and values of n = 10,000, 50,000, and 100,000. Data points were randomly generated using a uniform normal distribution (no duplicate points allowed), and 100 different randomly generated hyperplanes were used to perform a total of 900 different range reporting tests. The results of this experimentation are shown in Table 1. All of the software for this testing was written in C++ and run on a UNIX workstation (Sun SPARC 1000 with 640MB RAM (820MB of hard disk space available as virtual memory), two 50 MHz processors, running SunOS 5.3). Further details of the experiments are contained in [9].

Conclusions

A simple yet effective algorithm and data structure for dynamic k–d half–space range reporting has been presented. The dynamic update time of $O(k \log n)$ is the fastest that the authors are aware of. There is a trade–off with the worst case range reporting time which now becomes $O(k\,n)$. As suggested in [4], this leads one to ask if there is some lower bound on $P(n,k)S(n,k)Q(n,k)U(n,k)$, for P = preprocessing time, S = storage space, Q = time to perform range reporting for a given hyperplane, and U = time to perform a single update (insert or delete) operation. In our case, this product is $O(k^4 n^3 \log^2 n)$ (ignoring the time t for reporting). Other space/time trade–off studies [3], [4], [10], [11] have been carried out. Bronnimann et al [3] prove that if m units of storage are available, then the worst–case query time (for k–d half–space range reporting) must be on the order of $\dfrac{(n/\log n)^{1-(k-1)/k(k-1)}}{m^{1/k}}$. What trade–offs exist if dynamic operations are considered?

The algorithm and data structure reported here is likely to be useful in real applications. The times reported in the experimental results section indicate that our k–d half–space range reporting algorithm is fast, even for relatively large k and n. In addition, our algorithm provides very good support for highly dynamic applications due to the $O(k \log n)$ time for any single insertion

Table 1. Experimental results for k–d half–space range reporting (* indicates below measureable threshold).

		% of points in range		
		≤ 10%	40 to 60%	≥ 90%
k	n	average search time in seconds		
5	10,000	*	0.0054	0.017
	50,000	0.0008	0.045	0.090
	100,000	0.002	0.110	0.186
7	10,000	*	0.010	0.019
	50,000	0.001	0.055	0.1054
	100,000	0.0026	0.130	0.213
10	10,000	0.003	0.0097	0.022
	50,000	0.0044	0.0556	0.0994
	100,000	0.009	0.1066	0.1955

or deletion. The algorithm is easily adapted to queries with multiple hyperplanes. The average case performance of the algorithm presented here remains to be determined.

References

[1] Agarwal, P.K. and MatouÓek, J. "Dynamic half–space range reporting and its applications", *Algorithmica*, vol.13, 1995, pp.325–345.

[2] Bentley, J.L. and Maurer, H.A., "Efficient worst–case data structures for range searching", *Acta Informatica*, vol.13, no.2, 1980, pp. 155–168.

[3] Bronnimann, H., Chazelle, B. and Pach, J. "How hard is half–space range searching?", *Discrete & Computational Geometry*, vol.10, 1993, pp. 143–155.

[4] Chazelle, B. "Lower bounds for orthogonal range searching: I. The reporting case", Journal of the Association for Computing Machinery, vol.37, no.2, 1990, pp.200–212.

[5] Chazelle, B. and Preparata, F.P. "Halfspace range search: an algorithmic application of k–sets", *Discrete & Computational Geometry*, vol.1, 1986, pp. 83–93.

[6] MatouÓek, J. "Reporting points in halfspaces", *Computational Geometry: Theory and Applications*, vol.2, no.3, 1992, pp.169–186.

[7] Munro, J. Ian; Papadakis, Thomas; and Sedgewick, R., "Deterministic Skip Lists", In Proc. of the Third Annual Symposium on Discrete Algorithms, Orlando, Florida, January 27–29, 1992, pp.367–375.

[8] Nickerson, B.G. "Skip List Data Structures for Multidimensional Data", University of Maryland Institute for Advanced Computer Studies report UMIACS–TR–94–52, CS–TR–3262, Apr. 1994 (39 pages), available via anonymous ftp site "`ftp.cs.umd.edu`" file "`/pub/papers/TRs/3262.ps`".

[9] Pan, Y. "Investigation of a dynamic k–d search skip list requiring Θ(kn) space", UNB Faculty of Computer Science Technical Report TR97–116, 1997.

[10] Vaidya, P.M. "Space–time trade–offs for orthogonal range queries", *SIAM Journal of Computing*, vol.18, no.4, 1989, pp.748–758.

[11] Yao, A.C. "Space–time tradeoff for answering range queries", In Proc. of the 14th Annual ACM Symposium on the

Theory of Computing, San Francisco,
CA, May 5–7, 1982, pp.128–136.