1. Show that those MHRs in *T* not containing *X* will not be affected (which justifies the use of the function *ContainmentSearchO).*
2. Show that those MHRs in *T* containing *X* will not be maximal anymore (which justifies why they are deleted).
3. Show that the new MHRs must be inside the union of the MHRs found by *ContainmentSearchi)* (which justifies why the new MHR are generated only from these MHR found).
4. Show that the new MHRs must touch *X* and all possible MHRs that touch *X* will be reported (which justifies the way the new MHRs are constructed).
5. Show that no identical MHRs can be generated using this algorithm (which justifies why we do not check for duplicates).

A final point is that in our algorithm, the newly generated MHR is always contained within the MHR from which it is derived from, hence it is safe to discard any MHR that is not accepted by *BigEnough()* (by the definition of *BigEnough()).* With this, the correctness of the algorithm FindMHR is proved.

3.5 Run time complexity

Each insert and delete operation on T can be implemented in $O((\log m)^2)$ amortized time, where there are $O(m)$ number of MHRs stored in T. Containment search in T is done in $O(\log m + C)$ where there are $O(C)$ number of MHRs returned [Overmars and Leeuwen, 1982].

A loose bound on the total number of MHR (i.e., m) is $O(n^{2(k-1)})$ since a possible MHR can be constructed by choosing $2(k\text{-}1)$ points from the n points, to bound the k-1 dimensions, and the bounding points for the last dimension is simply decided from the geometry. We also assume that the number of bounding points per MHR is small, thus the operation to compute L'_j, U'_j, L''_j, L''_j is almost constant.

With this assumption, the algorithm runs in $O(n(\log m + C) + nC(\log m)^2 + nCk(\log m)^2 + nCk^2)$ which simplifies to $O(nCk(\log m)^2 + nCk^2)$. Since C is of $O(m)$ and m is of $O(n^{2k-2})$, the run time is simplified to $O(n^{2k-1}k^3(\log n)^2)$.

Although this complexity is high, in practice the number of *sufficiently large* MHRs is not many. Thus, computational cost is usually not a problem.

4 Experimental Results

The proposed algorithm is implemented in C++. Two sets of experiments have been conducted to test the efficiency and effectiveness of the algorithm in discovering the MHRs. The first set of experiments aims to show that the running time of the algorithm is reasonable. All the datasets are randomly generated. Some areas are intentionally left empty. But we have no knowledge about the existence of holes in other areas. Here, we only present the results from the 3D datasets because they are more representative of the real-life situations. Holes of more than 3D are hard to understand (interested reader may refer to [Ku *et al.*, 1997] for experimental results using higher dimensional datasets). Let *X*, *Y* and *Z* denote the 3 dimensions. The resolution of *X*, *Y* and *Z* is of an accuracy of up to 1/100. We run the first set of experiments on 3 different datasets. The lower and upper bounds for *X* and *Y* in all the three datasets are 0.00-23.00 and 0.00-17.00 respectively. The bounds for Z are 0.00-20.00, 0.00-70.00 and 0.00-150.00 respectively for the three datasets. The planted empty areas are of the same size, 9, 6 and 9 along *X, Y* and Z dimensions respectively. Table 1 summarizes the run time results of the three datasets (running on Digital Alpha 8400 under normal loading conditions). Note that *T* (which stores all the MHRs) is currently implemented as a linked list.

Table 1. Results of the first Set of Experiments.

Data set	No. of points	Running time *a*:*b* – *a* mins, *b* secs		No. of MHRs found	
		8–4–8	4–2–4	8–4–8	4–2–4
1	1427	0 : 02	0 : 06	85	2299
2	5064	0 : 15	1 : 05	162	6925
3	10702	1 : 10	6 : 17	260	13141

Column 1 indicates the dataset number. Column 2 shows the number of data points in each dataset. Column 3 gives the running times for finding all the MHRs that satisfy the minimal length requirements along the 3 dimensions. This column is further divided into two sub-columns. One of them shows the running times when the minimal lengths along *X, Y* and Z dimensions of the MHRs are 8, 4 and 8 respectively (i.e., the bounds used by *BigEnoughQ).* The other shows the running times when the minimal length is reduced by half along each dimension (i.e., 4-2-4). Column 4 gives the number of MHRs discovered by the algorithm in each of the two situations. From the table, we see that when the minimal size of the MHR decreases, the time taken to find all the sufficiently large MHRs increases. We also see that as the number of data points increases, the time taken to find all the large MHRs also increases. In general, however, the relationship between the number of data points and the time taken to find all the large MHRs is complex because there are other factors that play a role, e.g., the density and/or the distribution of the data points, and the number of large MHRs that exist in the dataset. In all experiments, the running times are reasonable. In spite of the large number of discovered MHRs (both planted and unknown), many of them actually represent the same general regions with slight variations (slightly different sets of bounding points). In practice, post-processing can be performed to extract those general empty regions.

In the second set of experiments, we use a real-life disease dataset. This dataset has 7 continuous attributes, and 713 data points. The algorithm is run 8 times using different attribute combinations, i.e., combining 2, or 3 or 4 attributes. The minimal size of the MHRs in each experiment is specified by a doctor. The running times of all the tests are within 1 or 2 seconds.

In these experiments, a number of interesting holes are discovered. For example, it is suspected that if the systolic blood pressure (SBP) of a subject is high, then his/her diastolic blood pressure (DBP) is also high. A hole is discovered in the region of high SBP and low DBP. This hole confirms the suspected fact. Some holes are quite unexpected.