CS 395 T
Modeling Biological Databases

Final Project
December 10, 2008

Guillermo Cabrera

Assessment of Excluded Middle Vantage Point Forests in MoBIoS

- **Introduction**

  The main task in this project focused on the assessment of the excluded middle idea mentioned in [YIA99] in terms of the Mobios system. Several steps have been taken in order to evaluate the speedup of such approach; further details and results are included below after a brief introduction to the project focus and background information.

  An excluded middle vp-forest structure is suggested in [YIA99], this forest is built by recursively removing the elements in the middle partition and instead putting them in a bucket. Once the tree is built, another tree is built from elements in the bucket in the same manner and so on, until the bucket is empty.

  The importance of this idea is reflected in the elimination of backtracking, in that we avoid the partial traversal of a tree structure, because elements that are located near the middle of the distribution values were removed; thus, only corner partitions are left in each internal node of the tree.

  The current code in Mobios uses a multivantage point tree structure as defined in [BOZ99] to build indexes; the tree's layout can be modified by a series of parameters used in building the index. Such parameters include: Pivots per node, single pivot fanout, partition method to use, etc. Some of these options will be used in measuring the middle partitions for the sample data sets included in Mobios, which constitute a control work-set useful in comparing any results from the project.

  In this project we focus in obtaining two types of results; first we need to find *m* (proportion of points in middle partitions) for the Mobios workloads, Yianilos uses this parameter to discuss performance in an "idealized tree" case, *m* determines the number of trees that will conform the forest, time to build it and the search time for a given query. Given this first result, the second part consists on devising an implantation following the main algorithm in [YIA99], this will yield results that will let us compare the build and query performance of this approach and whether it is a viable alternative to be incorporated into the Mobios system.

  The following sections expand on how these two types of results were obtained, key ideas, problems in implementing, future work and other details pertaining to the implantation of the excluded middle vp-tree structure.

- **Middle Proportion on Workloads**

  The examples presented by the Mobios page cover an ample spectrum of data types (sequence, spectra, and vector) to be indexed; moreover, the included build and query commands give grounds for comparison with experimental results. Yet, in finding the middle proportion (*m*) we are interested in the distribution of the data in the metric space; this will give a clear picture as how build and query performance will change if an excluded middle vp-tree structure were used.

  - Process

    1. For each partition in the root node
       1.1 Calculate distance of pivot to each element in partition
       1.2 Save to file
    2. Compute distribution of all distances

    Step 1 of this process happens in the *mobios.index.VPIndex* class; in the load method we first start by selecting the pivots for the data set to be indexed and we create an internal node

(root) which involves an initial partition of the data. After this initial partition each one of them will be recursively partitioned until data elements are written to leaf nodes. Changes in the build command (only 1 pivot and fanout of 3) have been made in order to produce a tree layout with 3 partitions per node, thus, making it trivial to compute the distribution based on the above process. The distribution graphs as well as information relevant to the data set are included below.

- Results

DNA Sequence

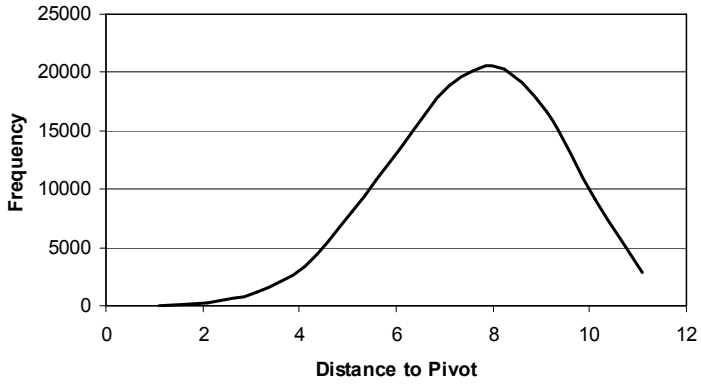

Figure 1: DNA Sequence histogram

Additional Information			
Max Dist.	11	**Total Points**	94382
Min Dist.	1	**Points in L**	28678
Mean	1.765	**Points in M**	39327
Std. Dev	7.499	**Points in R**	26377

Table 1: DNA Sequence information

Peptide

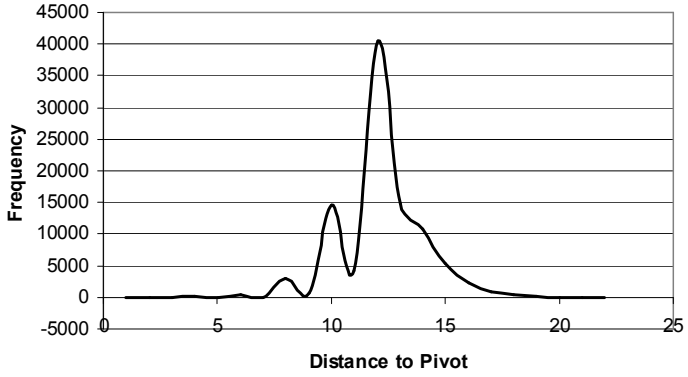

Figure 2: Peptide histogram

Additional Information			
Max Dist.	22	**Total Points**	98064
Min Dist.	2	**Points in L**	34422
Mean	12.179	**Points in M**	40516
Std. Dev	1.819	**Points in R**	23126

Table 2: Peptide information

Spectra (msms)

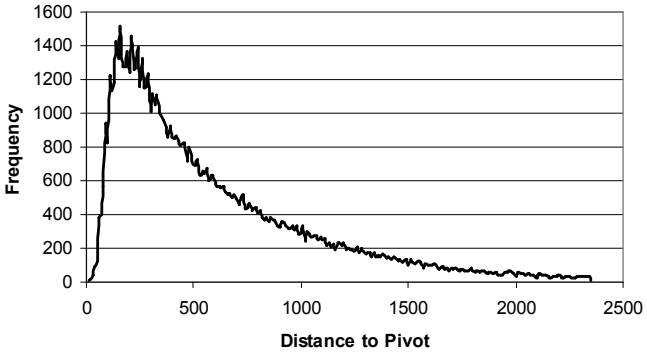

Figure 3: Spectra histogram

Additional Information			
Max Dist.	2342.734	**Total Points**	84584
Min Dist.	1.445	**Points in L**	28195
Mean	589.889	**Points in M**	28195
Std. Dev	465.644	**Points in R**	28194

Table 3: Spectra information

Vector Uniform

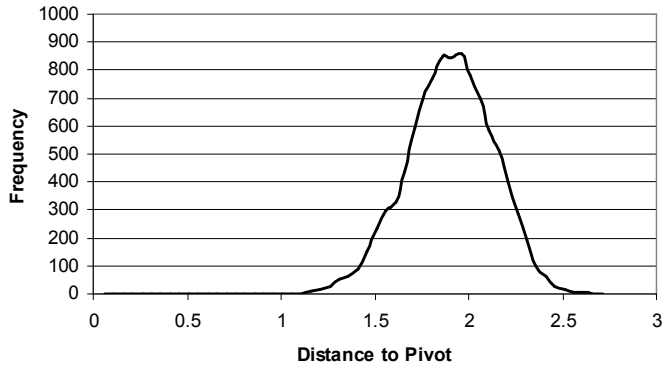

Figure 4: Vector uniform histogram

Additional Information			
Max Dist.	2.664	**Total Points**	9999
Min Dist.	0.941	**Points in L**	3333
Mean	1.876	**Points in M**	3333
Std. Dev	0.231	**Points in R**	3333

Table 4: Vector uniform information

Vector Random

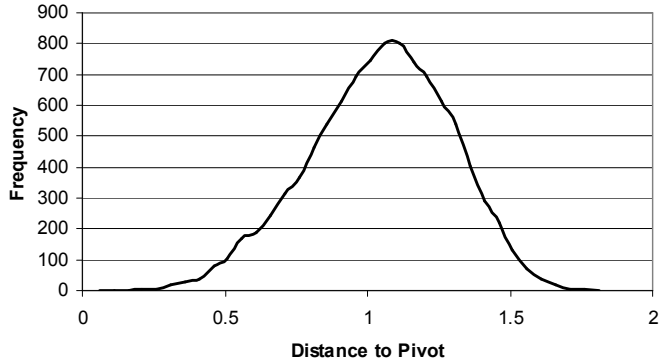

Figure 5: Vector random histogram

Additional Information			
Max Dist.	1.793	**Total Points**	9999
Min Dist.	0.155	**Points in L**	3333
Mean	1.019	**Points in M**	3333
Std. Dev	0.251	**Points in R**	3333

Table 5: Vector random information

Workload	Max Dist.	Min Dist.	Mean	Std. Dev.	Total Points	Points in L	Points in M	Points in R	m
DNA	11	1	1.765	7.499	94382	28678	39327	26377	**0.417**
Peptide	22	2	12.179	1.819	98064	34422	40516	23126	**0.413**
Msms	2342.734	1.445	589.889	65.644	84584	28195	28195	28194	**0.333**
VectorU	2.664	0.941	1.876	0.231	9999	3333	3333	3333	**0.333**
VectorR	1.793	0.155	1.019	0.251	9999	3333	3333	3333	**0.333**

Table 6: Summary of all workloads including *m*

- Discussion

The above data sets which represent real workloads all have different distributions, it is interesting to note that even if we use a “balanced” method to divide the number of elements at each node, it is not the case that we will have the same number of elements in each partition; an assumption I had at the beginning of the project. Thus, in these two cases (dna and peptide) the middle partition proportion is higher than that of the left and right subtrees.

A higher concentration on elements in the middle partitions translates to more trees in the forest, since these middle elements are added to the bucket which is later used to construct a new tree. We can further asses this idea by referring to the formulas provided in proposition 1 of [YIA99]:

From proposition 1 we define $\rho = 1/(1 - \log_2 (1 - m))$

Then:

* There are $\Theta (N^{1-\rho})$ trees in the forest having maximum depth $\Theta (\log N)$

In the case of the dna workload $\rho = 0.810$ which means that for this workload we would have $\Theta (N^{0.19})$ trees. Further calculations can be made in terms of search time and build time of the index. The key factor in implementing the excluded middle idea lies in how many trees would be constructed from data sets; we don't want to keep this number to a minimum.

- **Excluded Middle Implementation**

As mentioned earlier the Mobios system produces a single mvp-tree for the index being created, this contrasts with our approach of creating multiple trees to form a forest. So, as an initial stage we have made modification to existing classes to be able to support the construction of multiple trees, yet, making in the end delivering a single tree structure as output of the build process.

The main idea is to build the first tree in the manner described in [YIA99] once we are done creating this tree we save its root node and pointer to root in a list, we then process the next tree from elements in a bucket (previously added from construction of first tree). We will continue to do so until there are no more elements to process. In the end we will create a new root (**super root**) that has a pointer to all the previously created root nodes. In the image below, the gray nodes represent root nodes of sub-trees 1,2, k. Sub-trees 2 to k were formed from elements in the bucket.

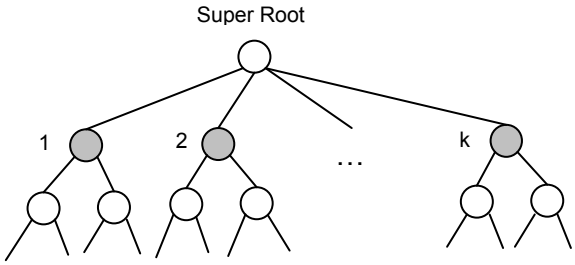

Figure 6: Single tree representing a forest

- Algorithm

1. Create tree
   1.1 For each internal node of the tree
      1.1.1 Remove middle partition and assign elements to BUCKET
   1.2 Write root of tree
   1.3 Save pointer to this root and root node to a LIST
2. If BUCKET is not empty
   2.1 Go to step 1, use elements in BUCKET as input to tree
3. Create new internal node (super root)
   3.1 Assign all previous root addresses in LIST as children of this super root
4. Return pointer to super root

- Details of Implementation

Implementation is based on a new partition method included in the PartitionMethods class. The partition method handles the division of the data set into three partitions according to the idealized version in [YIA99]. Later, the middle partition is removed and all its elements are added to the bucket, thus, leaving the particular node being partitioned with only a left and

right partitions.

The load method in VPIndex will be called recursively after the building of a single tree, and its input will be the elements in the bucket. After all k trees have been built, creation of the super root involves three main tasks: Setting its correct upper and lower ranges for each child, the pivots, and the child addresses; all of this information is provided by the member variables mentioned above.

Two classes have been modified in order to implement the above algorithm, they include: VPIndex, PartitionMethods. In addition, the changes require that an index be built with the following flags: "-dpm EXCLUDEDTEST –f 2 –p 1", we are using a single vantage point to partition the data, are building a forest of B-trees (since we are removing the middle partition), and using a new partition method to handle the removal of middle partitions. New transient member variables were added to VPIndex to account for the "bucket" and "list" mentioned in the algorithm.

- Results

Testing was divided into two areas, building and querying the index. For the first we measured the number of nodes in the index (size of the Object I/O Manager) and the number of distance calculations. And querying the index, for the second we also measured the number of distance calculations and measured the number of nodes visited.

Furthermore, each of the five workloads was built in three modes:

- DEF: Using the default code with no modifications and the build and query commands as defined in the scripts.
- MOD: Same as above with the only difference that we use only one vp.
- ET: This mode makes use of the implementation changes, thus, three flags are passed for building an index: -dpm EXCLUDEDTEST –f 2 –p 1.

Below I include the output of the above mentioned attributes for building the five workloads under the three different modes. For querying the indexes I have chosen three workloads that are representative of the behavior shown at build time.

Build

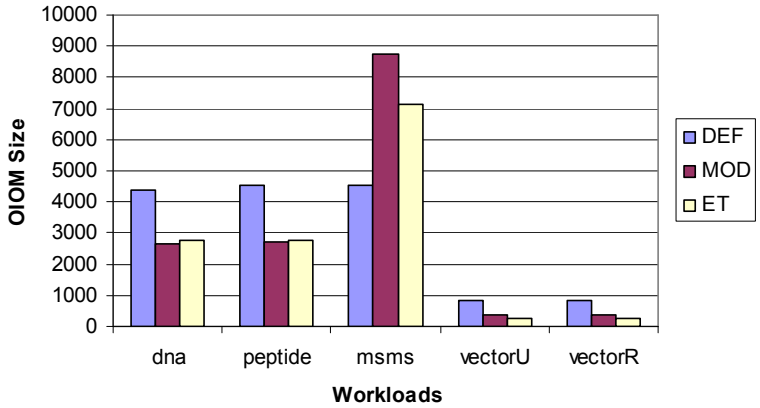

Figure 7: Number of nodes created during build

The Object I/O Manager is called every time we write an node to disk, thus, the size reflects the number of nodes that have been written out. For all of the workloads except for msms, the excluded middle implementation (ET) writes **fewer nodes** than the default (DEF) implementation.

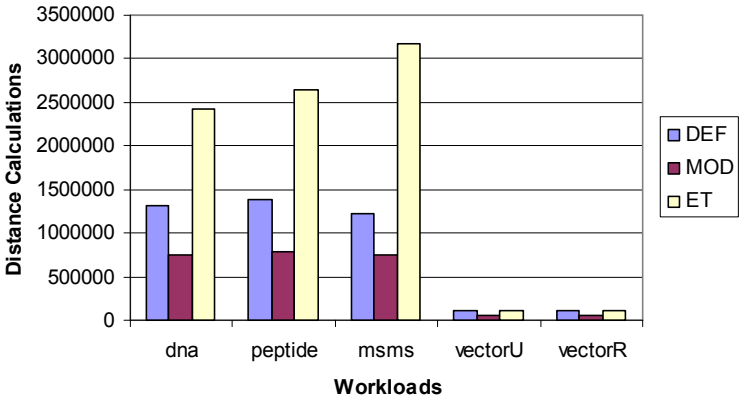

Figure 8: Distance calculations at build time

One of the reasons we see a higher distance calculation bar for the excluded middle implementation (ET) is because distance is **recalculated** for data elements that are added to a bucket. When the next tree in the forest is built the new data set from the bucket will have a new pivot and distances from this pivot need to be calculated. It is interesting to see the difference for the first 3 workloads is almost double the default case, yet for vector workloads it remains approximately the same.

Query

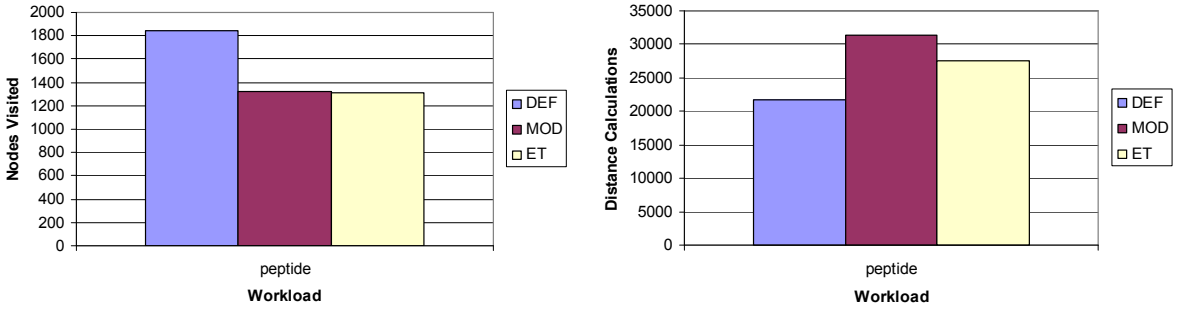

Figure 9: Query performance of peptide workload

At query time, we can see that for this specific workload we do **visit less nodes** in order to get the elements that fall under the radius specified in the query. This translates to less **I/O** operations in retrieving the result. Yet, at the same time we see a slightly higher number of distance calculations in comparing ET with DEF.

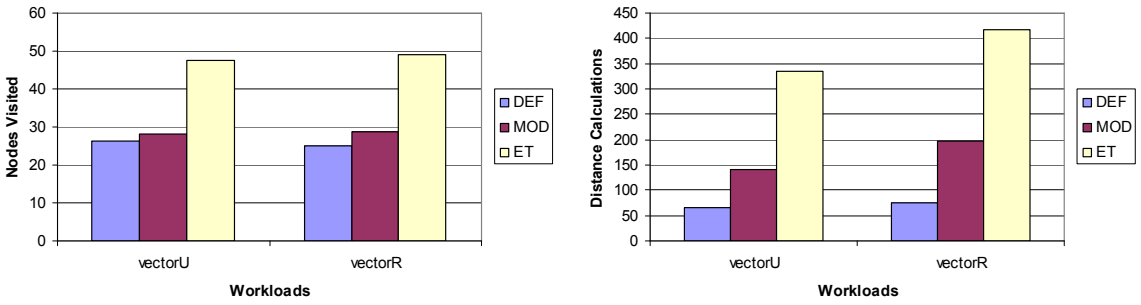

Figure 10: Query performance of vector workloads

A contrast to the above workload happens with the vector workloads where both nodes visited and distance calculations in ET are significantly elevated from DEF and MOD cases. This behavior could be the result of the curse of dimensionality given that the vector workloads are in 20 dimensions.

- **Conclusion and Future Work**

  Results obtained for the first section on finding the middle proportion of Mobios workloads were satisfactory. Yet, the second section involved with results of the implementation did not show an evident pattern of speedup when compared with the default case; there are however some limitations in the current implementation that prevent us from obtaining the improved performance results.

  One major limitation is the number of pivots we are working with, currently we work with a single pivot per node, thus, there is a high probability that searches visit at least two partitions at each level of the B-tree. Weijia Xu (former member of MoBIos group) suggested we consider two pivots per level; this would enable the selection of corner partitions while excluding points that fall in the middle from the corners.

  Furthermore, the partition method used in the excluded middle implementation is currently built upon the a version of the “balanced” partition method where sizes for each partition are approximately the same size. In our implementation we have followed the idealized version of the excluded middle vp-tree structure, thus, we are not taking into consideration the radius as input in the index building process. The radius specified by the user represents a common search query, thus, in building the index we should optimize for future queries involving radius of that length. Yianilos suggests variable central cuts be made using a diameter of 2r.

- **References**

  - [BOZ99] Bozkaya and Ozsoyoglu "Indexing Large Metric Spaces for Similarity Search Queries"
  - [MAO06] Mao, et al. "MoBIoS Index: Support Distance-Based Queries in Bioinformatics"
  - MoBIos, MoBIos website, http://www.cs.utexas.edu/~mobios.
  - [YIA99] Yianilos, Peter "Excluded Middle Vantage Point Forests for Nearest Neighbor Search"