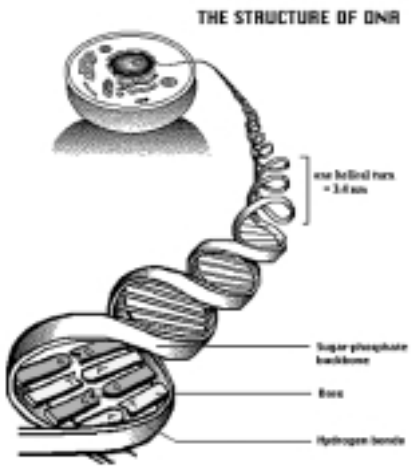
DNA Indexing

Presented by
Cheng Lok Lam

Contents

- DNA
- Suffix Trees
- Hunt's Paper in VLDB 2001
- Proposed Algorithms
- Experiments
- Conclusion

The Structure of DNA

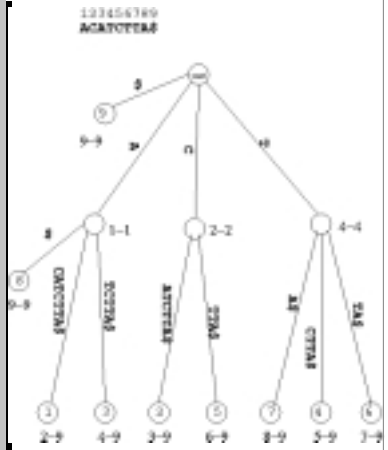


More about DNA

- A DNA sequence can be abstracted as a sequence of 4 basic characters -- A, C, G and T.
- In human, the total length of DNA is expected to be 3.2G characters.
- DNA sequences have high entropy and can be threated as pseudo-random.

Suffix Tree

- A suffix Tree *T* of a string *S* with *n* characters is a rooted directed tree with exactly *n* leaves numbered 1 to *n.*
- All internal nodes in *T* has at least 2 children.
- Each edge in *T* is labeled with a nonempty substring of *S.*
- No two edges out of a node can have edge-labels beginning with the same character.
- Concatenation of the edge-labels on the path from root to the node *i* is *S[i..n].*

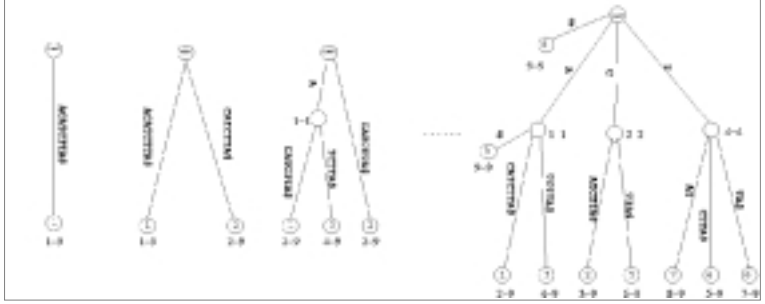


Advantages of Suffix Tree

- O(*n*) building time
- O(*n*) storage
- O(*l* + *h*) searching time, (*l* is query length and *h* is number of hits)

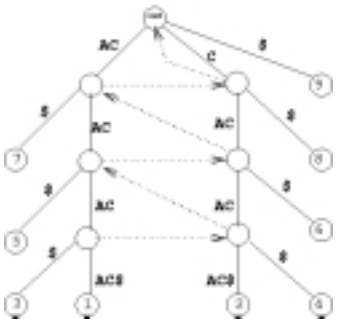
Building Suffix Trees (1)

- Naïve Algorithm -- $O(n^2)$
  - Start with a tree contains root and a leave node 1.
  - Adding i^{th} suffix to the tree by search the longest common prefix of $(i-1)^{th}$ in the tree.



Building Suffix Trees (2)

- Using suffix links - O(n)
  - A suffix link exists for each internal node and points from node indexing *aw* to node indexing *w*
  - This accelerates finding next longest common prefix



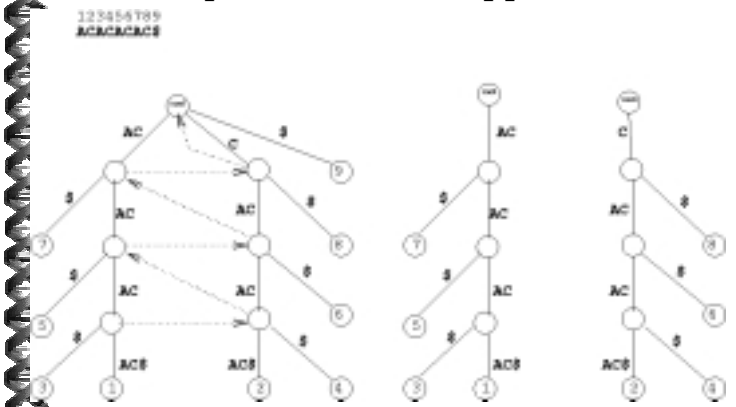
DNA + Suffix Tree

- DNA sequences are very large
- The best space efficient algorithm by S. Kurtz is about 13 bytes per DNA character
- 3.2 G $\times$ 13 = 41.6G
- Impossible to build the suffix tree on Main Memory only

Hunt's Approach

- Hunt's paper, *A Database Index to Large Biological Sequences*, in Proc. VLDB 2001
- give up using suffix links
- Partition the suffix tree into approximately equal size sub-trees such that each tree is small enough to build within main memory only.
- Partition based on the prefix of each suffix. e.g. Sub-tree$_1$ only contains the suffixes starting with " AA", sub-tree$_2$ only contains suffixs starting with " AB"....

Example of Hunt's Approach



Disadvantages of Hunt's Approach

- Cannot using linear time tree building algorithm. Can only using naïve algorithm.
- Required to load the whole sequence into memory. If the size of the sequence is larger than that of memory, the approach doesn't work.

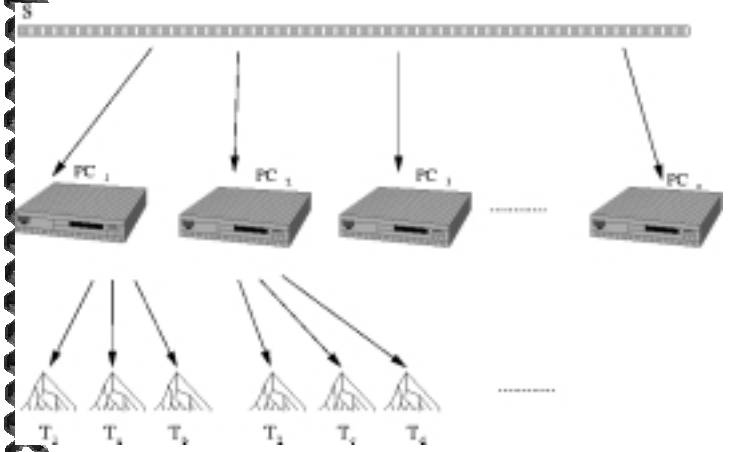
Our approach

- Use PC clusters to build index and search in parallel.
- Improve the index building and searching time without memory bottle-neck.
- We proposed 3 different algorithms for tree building in clusters
  - Algorithm TP -- (Tree partition)
  - Algorithm DP -- (Data partition)
  - Algorithm H -- (Hybrid)

Algorithm TP -- Tree Partition

- The idea is come from Hunt's approach.
- Assuming there are N PCs in the cluster.
- Each PC can support to build a suffix tree with M suffixes
- So we partition it into L/M partitions where L is the length of the sequence
- During tree building stage, the partitions will be assigned to a PC in round robin fashion. e.g. PC_1 build P_1, PC_2 build P_2 ....

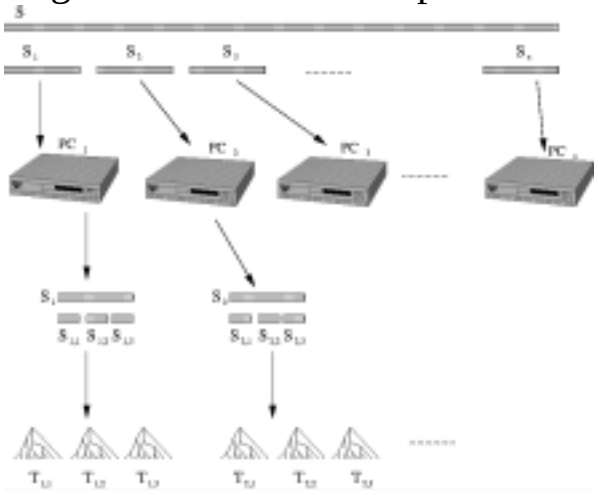
Algorithm TP



Algorithm DP -- data partition

- The main idea is to partition the sequence S in to smaller subsequences so the suffix tree of each subsequence can be built on main memory only to avoid IO.
- O(n) algorithm can be used for building suffix tree for each subsequences.
- Assuming there are N PCs in the clusters. First, cut S in to N subsequeces and each PC get one of the subsequences.
- The i^{th} PC gets the subsequence S'_i.
- If the suffix tree of S'_i is too large, S'_i will be further divided into smaller subsequences.

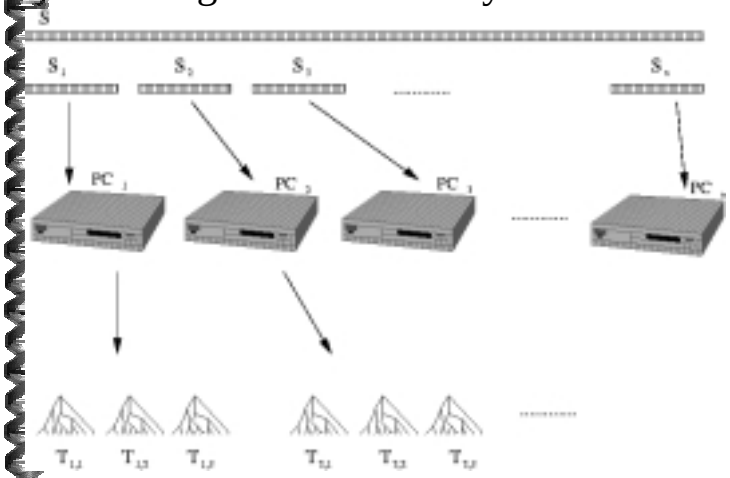
Algorithm DP -- data partition



Algorithm H -- Hybrid

- It is a mixed version of algorithm TP and algorithm DP
- First, similar to algorithm DP, divide the sequence S in to N subsequences and each PC get one of the subsequences.
- In each PC, it used Hunt's approach to partition the suffix tree of S_i, so that each tree partition can be build in the main memory.

Algorithm H -- Hybrid



Cross Boundary Cases

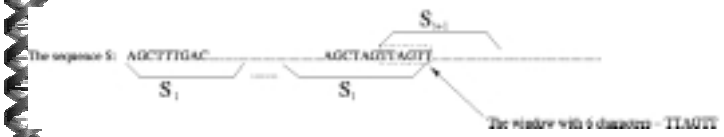
- In algorithm H and DP, we cut the sequence S into smaller subsequences. We need to handle the cross boundary cases.

Cross Boundary Cases

- The prefix of the query string may match the suffix of S_i. We define this match as **prefix-suffix hit** (PSH). The length of the matched part is called **PSH length**.
- If there exists PSH in S_i, we need to search whether the remaining characters match the prefix of next sequence S_{i+1}.

Partitioning with Windows

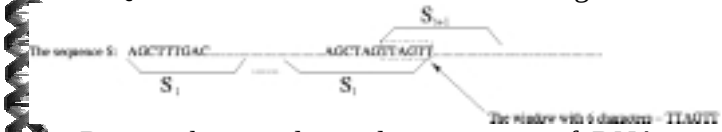
- To reduce the probability of the cross boundary cases, a small overlap of partition is introduced.



- The part of overlap is called **window**.
- Length of the overlap is called **window size**.
- When window exists, we don't need to consider all PHS with length less than the window size

Partitioning with Windows

- If Q = " AGTTACTTCTTT" PSH length = 4
- If Q = " AGTTAGTTCTTT" PSH length = 8



- Due to the pseudo-random property of DNA sequences, a small window size can greatly reduce the number of cross boundary case.
- $E < (q\text{-}x\text{-}1)(1/c)^{x+1}$,
  - E is average number of PHS with length $\geq x$ at a cross boundary with window size of x and a query with length q.
  - c is the character set size. It is 4 in DNA.

Experiment Environment

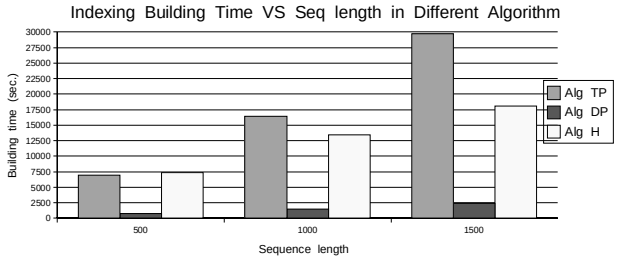
- 32 nodes
- 2 X PIII 1GHz CPU per node
- 2G RAM per node
- RedHat 7.2
- Linux 2.4.7 SMP kernel
- MPICH 1.2.1

Experiment 1 (Index Building)

- Building index for 500M, 1000M and 1500M DNA sequences with the three algorithms
- The data is get from National Center for Biotechnology Information ftp://ncbi.nlm.nih.gov/genomes/H_sapiens
- Using 9 nodes, 1 for master node, 8 calculation nodes.
- We only use 1.7G RAM for storing the building tree and the sequences.
- The window size is set to 10

Experiment 1 (Index Building)

	Alg TP		Alg DP		Alg H	
SEQ Length	Total number of tree	Building time	Tree per Node	Building time	Tree per Node	Building time
500M	8	6980s	1	733s	1	7375s
1000M	16	16389s	2	1514s	2	13443s
1500M	32	29765s	3	2394s	3	18085s

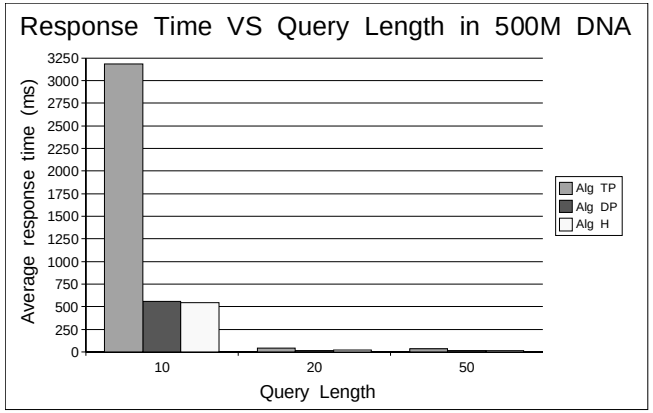


Experiment 2 (Searching)

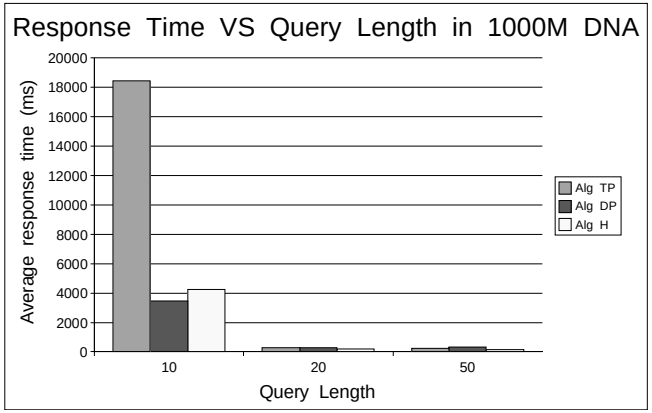
- Using the indexes built in experiment 1 for searching.
- Different query length (10, 20, 50) were tested.
- For each query length and algorithm, we got average time from issuing a batch of 1000 queries.
- We found that there is no PHS in any queries for a window size of 10

	Number of Hit of Diffreent Query length		
	10	**20**	**50**
500M	1336608	6305	3150
1000M	2642128	14947	6011
1500M	4021869	17945	8003

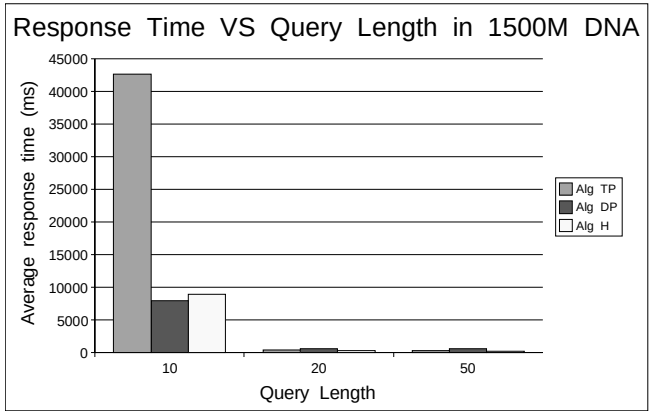
Experiment 2 (Searching)



Experiment 2 (Searching)



Experiment 2 (Searching)



PSH Rate

- The probability of PSH is very low
- The following table shows the number of PSH for different query length, sequence length and algorithm in a batch of 1000 queries.

	Alg DP		Alg H	
Seq Length(M)	Q20	Q50	Q20	Q50
500	0	0	0	0
1000	0	0	0	0
1500	1	1	1	0

Conclusion

- Alg. DP is the best in term of index building time
- The query response time of Algorithm TP is good for small number of hits, while Alg. DP and Alg. H is good for large number of hits.
- A window can greatly reduce the cross boundary effect in Alg. DP and Alg. H.

Q&A