Distributed Suffix Trees on Apache Hama

Reid Horton[1], Joohyun Kim[2]
[1]Department of Computer Science, Stony Brook University, Stony Brook, NY 11794
[2]Center for Computation and Technology, Louisiana State University, Baton Rouge, LA 70803

Introduction

Suffix trees are important data structures that reduce the running times of several pattern matching algorithms used to processing large amounts of genome information.

We have implemented an analogous data structure that can be spread out across many computers, allowing one to work with larger data sets than can be handled by a single machine.

Our long term goal is to develop a transcriptome assembler that can run in the cloud by combining this distributed data structure with existing genome assembly software.

Background

Suffix trees work by storing every ending of a sequence of characters (see figures 1 & 2).

In a distributed suffix tree, several computers work together to store all of these substrings. Each computer is assigned a different subset of suffixes, and a sparse suffix tree [2] is constructed using only those suffixes (figure 3).

Figure 1

G	C	A	C	A	T	T	G	A	$
	C	A	C	A	T	T	G	A	$
		A	C	A	T	T	G	A	$
			C	A	T	T	G	A	$
				A	T	T	G	A	$
					T	T	G	A	$
						T	G	A	$
							G	A	$
								A	$

Here we have every suffix of the string GCACATTGA$. A suffix tree for this string needs to represent all of these substrings with a path in the tree from the root to a leaf.

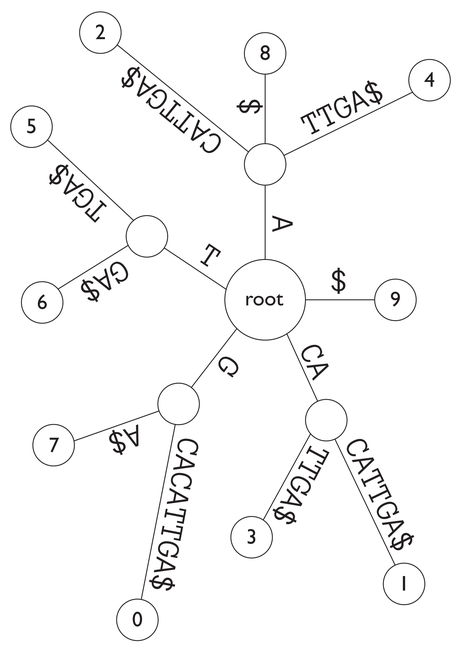
Development

Constructing a distributed suffix tree requires modifying standard suffix tree construction algorithms if we want it to work with very long strings – the entire human genome, for example. Our C++ implementation uses an already known approach [1].

Hama is a distributed computing framework that runs on top of Hadoop, and it allows us to manage efficient communication between compute nodes.

While sparse suffix trees can actually be constructed without communication, we hope to leverage more of Hama's power while developing the assembly program.

Figure 2



This is the suffix tree for the string GCACATTGA$. Every substring from figure 1 is contained in this tree as a path from root to a leaf. The number inside each leaf is the index (in the original string) of the suffix that ends at that leaf.

Results

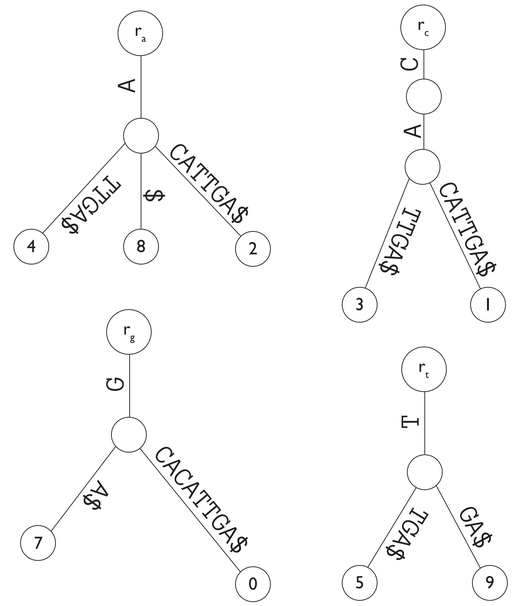
In an ideal world, distributed suffix trees can be built in linear time – an input twice as big should only take twice as long to process.

Preliminary tests using real world genome data show that our construction algorithm is exhibiting *mostly* linear performance (figure 4).

More testing is required to determine if our current implementation is practical for much larger data sets, such as the entire human transcriptome.

A very important property of our implementation is that by using more specific prefixes [1], the data can be broken up into arbitrarily small pieces (see figure 3).

Figure 3



These four sparse suffix trees each store suffixes beginning with a certain sequence of characters. Longer prefixes can be used to construct smaller trees, which allows the data to be spread out further [1].

Summary

Previous studies have come up with algorithms for distributed suffix trees, but they are not currently being utilized by the bioinformatics community. Suffix arrays (not distributed) are still the data structure of choice – in total, they require far less memory than any type of suffix tree.

However, as the amount of collected information continues to increase, more data will arise than any one machine can handle – this is where a distributed solution is most useful.

Future Work

The next step for our transcriptome assembly program is to implement inexact string matching (string matching with tolerance for mismatches) with distributed suffix trees.

We plan to interface distributed suffix trees running on Apache Hama with SGA, an existing genome assembly program (also written in C++).

The program will run on clusters of Amazon EC2 virtual machines, and will be accessible as a web application. Ideally, the application will dynamically allocate cloud compute resources based on the size of the given data set.

Figure 4

This graph shows distributed suffix tree construction times using actual DNA sequencing data. The tests were run on a 3-node cluster using Amazon EC2 m4.xlarge instances.

References

[1] Clifford, R., & Sergot, M. (2003, January). Distributed and paged suffix trees for large genetic databases. In Combinatorial Pattern Matching (pp. 70-82). Springer Berlin Heidelberg.

[2] Kärkkäinen, J., & Ukkonen, E. (1996). Sparse suffix trees. In Computing and Combinatorics (pp. 219-230). Springer Berlin Heidelberg.

Acknowledgements

This material is based upon work supported by the National Science Foundation under award OCI-W1263236 with additional support from the Center for Computation & Technology at Louisiana State University.

LSU | Center for Computation & Technology

Stony Brook University