ABSTRACT

OPTIMIZING APPROACHES FOR SENSITIVE, HIGH PERFORMANCE CLUSTERING OF GENE EXPRESSIONS

by James C. Moler

This thesis presents several new algorithmic approaches to the problem of clustering conventional ESTs and high throughput gene expression data, which are implemented in the software tool PEACE. The d^2 algorithm for sequence comparison is improved and enhanced with a novel two-pass extension, and a minimum spanning tree-based algorithm is used to cluster ESTs, providing an efficient and accurate solution. Furthermore, in order to address the unique challenges of high throughput sequencing technologies such as 454, Illumina and SOLiD sequencing, an adaptive d^2 algorithm is introduced to handle variations in fragment length. The resulting tool compares favorably with other leading tools in the literature, including WCD, CAP3, and TGICL, on both EST and next-generation sequencing (NGS) data.

OPTIMIZING APPROACHES FOR SENSITIVE, HIGH PERFORMANCE CLUSTERING OF GENE EXPRESSIONS

A Thesis

Submitted to the
Faculty of Miami University
in partial fulfillment of
the requirements for the degree of
Master of Science
Department of Computer Science and Software Engineering
by

James Clark Moler III
Miami University
Oxford, Ohio
2010

Advisor_____
Dr. John Karro

Reader_____
Dr. Dhananjai Rao

Reader_____
Dr. Mufit Ozden

Table of Contents

List of Figures

ACKNOWLEDGEMENTS

I would first like to thank Dr. Dhananjai Rao, both for his work in designing and implementing the PEACE software system, and for the guidance and advice he has given to me personally as a computer scientist and software developer. I would also like to acknowledge Dr. Mufit Ozden and Dr. Chun Liang for their ideas, collaboration, and the assistance they have given me throughout the PEACE project, this thesis, and my tenure as a graduate student at Miami. I would be remiss in not thanking Dr. Alton Sanders, who encouraged me to apply to graduate school in the first place. Finally, I want to thank my advisor, Dr. John Karro, for introducing me to bioinformatics, guiding me throughout my pursuit of my degree, advising me on applying to graduate programs and setting a model example for me to follow professionally and personally. Thank you for everything you have done.

1. Introduction

In the field of genome biology, analysis of the transcriptome – the set of expressed genes – is a branch of study with many applications including discovering genes, identifying gene structure, characterizing single nucleotide polymorphisms (SNPs), and studying the mechanism of alternative splicing. Historically, a significant amount of information about transcriptomes comes from expressed sequence tags, or ESTs – short, low-quality nucleotide sequences that can be generated in large numbers at a low cost. While ESTs remain a very relevant source of biological information, recent advances in high-throughput sequencing and the technique of RNA-Seq (applying high-throughput sequencing technologies to sequence cDNA) have made it possible to generate incredibly vast numbers of sequence fragments from transcript data, through technologies such as 454 Life Sciences' 454 Sequencing [1], Illumina's Genome Analyzer [2], and ABI's Solid Sequencing [3]. Generating fragments by these methods is much less expensive and provides a much greater volume of data, as well as a much deeper sequencing, but analysis of the data post-sequencing becomes more difficult due to the volume of data as well as the fact that reads are shorter than conventional Sanger-sequenced ESTs – for example, around 75 base pairs in length for Illumina reads.

In the application of gene discovery, there are four main steps: cleaning, clustering, assembly and annotation. Cleaning is a pre-processing step that involves taking the raw sequencer output and performing specific techniques to improve the data quality. For example, depending on the sequencing technology used, one can find low-quality segments on the ends of the reads, as well as stutters (long strings of repeats) within the reads. Since modern sequencing technology provides a quality value along with each base indicating the confidence in that particular base's correctness, low-quality regions can be identified. Thus, in the cleaning step, pre-processing tools and algorithms can be used to trim the low-quality ends and/or mask repeats or low-quality regions in the reads.

The clustering step concerns the partitioning of the set of sequences into separate groups, or clusters, such that each group contains only the sequences read from the same gene transcript.

This thesis will primarily deal with that step, with tangential discussions of other steps where necessary.

In the assembly step, the sets of sequences produced by clustering are used to reconstruct the transcript sequences from which the reads were originally generated, typically through the use of sequence alignment techniques. The sequences generated in this step are referred to as contigs. Finally, in the annotation step, biological information is attached to the sequences. The annotation step is beyond the scope of this thesis and will not be discussed here.

It is important to note that in some existing tools, algorithms and pipelines, the clustering and assembly steps are performed together. This will be discussed further in the related works section. In the work performed in this thesis, the clustering step was done separately from the assembly step. The merits and consequences of this approach will also be discussed later in the paper.

2. Background and Related Work

2.1 Background

Expressed sequence tags are generated from gene sequences, which are composed of four different bases: adenine, cytosine, guanine, and thymine. These bases are commonly denoted with the letters A, C, G, and T. For the purpose of quantitative analysis, a sequence can be considered a string over the alphabet {A, C, G, T}. ESTs are subsequences of genes, generated using a shotgun approach that results in a large number of ESTs derived from many different genes, all in the same set of data. Many of these ESTs overlap with other ESTs in the data set, and by finding the overlaps we can perform EST clustering, grouping the ESTs by gene. This is an essential step towards assembly.

The process of clustering ESTs naturally requires some means of comparing individual ESTs and finding the overlaps between them. Since EST sequences can be defined as strings, string similarity or string distance measures can be used for performing these comparisons in clustering and assembly tools. A string similarity measure takes as input two strings (ESTs in this case) and outputs a numerical value that approximates how similar the two strings are. Higher numbers mean that the strings are more similar. A classic example of a string similarity measure with applications in biology is the Smith-Waterman algorithm [4]. The Smith-Waterman algorithm is ideal for small data sets and is a fairly fast algorithm, but as the problem size increases (with EST data sets in the hundreds of thousands), Smith-Waterman is not fast enough to perform all of the pairwise comparisons that are necessary.

A string distance measure is the opposite of a similarity measure. It outputs a value indicating some measure of the distance between the two strings. In this case, a lower number means that the strings are more similar.

2.2 The d^2 algorithm

d^2 [5] is an algorithm for string distance originally developed for database search, and validated as a method for comparing genomic sequences [6]. d^2 is based on the comparison of word frequencies within windows of set length on two sequences. Every window on one sequence is compared with every window on the other sequence, and the minimum Euclidean distance between word frequencies among all pairs of windows is computed. This becomes the distance between the two sequences. d^2 is a symmetric distance measure, but does not adhere to the triangle inequality and thus is not a metric.

The d^2 algorithm has a quadratic runtime bound, but it can be computed more efficiently through the use of an incremental approach [7]. In this approach, the windows on each sequence are initialized only once. Then, to compare every possible window on every sequence as required, we slide the windows along the sequences one character at a time. When the window is shifted by one character, there are only two changes in word frequencies: the word at the beginning of the old window drops out of the window, and a word at the end of the new window is added; the resulting change in distance can be computed by a simple formula [7]. This incremental approach greatly improves the performance of the algorithm in practice (though its asymptotic performance is unchanged). Because of this improved performance, the d^2 algorithm is much faster than alignment-based sequence comparison approaches, and it attains results that are comparable in quality as shown by the following tools.

2.3 Related tools

There exist many tools for expressed sequence tag clustering, assembly and consensus sequence generation—an extensive overview of these tools can be found in *Nagaraj et al.* [8]. This section will focus on works with particular relevance to the research being proposed.

d^2 was first used for EST clustering in the algorithm d2_cluster [9]. d2_cluster is an agglomerative method of clustering, meaning every EST is initially in its own cluster, and clusters are successively compared and merged if sufficiently similar, as determined by application of the d^2 string distance measure. This clustering algorithm groups by transitive closure, as two dissimilar ESTs A and B will end up in the same cluster if there exists some similar EST C having sufficient similarity to both A and B.

More recently, d^2 is used in the EST clustering tool WCD [10]. WCD's overall strategy for clustering is similar to that of d2_cluster—WCD uses the same agglomerative, transitive closure method of forming the clusters, and uses d^2 for the sequence comparisons. However, to accompany the efficient implementation of d^2 described above, WCD introduces a set of linear heuristics for sequence comparison that eliminates around 99% of costly d^2 pairwise comparisons and does not noticeably impact the quality of the resulting clustering [11]. Additionally, WCD supports parallelization using the MPI (Message Passing Interface) standard and has clustered data sets of more than 600,000 ESTs.

The heuristics introduced by WCD are the primary source of its speedup. The first heuristic used, called the *uv* heuristic, functions by building a table of words of length *v* in the reference sequence. The heuristic then "skips" along the sequence to be compared, looking for at least *u* occurrences of words of length *v* that match words in the reference sequence. This simple heuristic filters out an impressive number of non-matching sequences. If the *uv* heuristic returns a positive result for the two sequences (meaning they should be analyzed further), WCD runs the *tv* heuristic, which searches the entirety of the comparison sequence for a 100-base window containing *t* words of length *v* that match words occurring anywhere in the reference sequence. This heuristic roughly approximates the behavior of the d^2 algorithm, but by allowing the matching words to occur anywhere in the reference sequence, it can be executed in linear time. Together, these heuristics allow WCD to cluster large data sets at impressive speeds.

PaCE [12] is a clustering tool, supporting serial and parallel modes, that uses an agglomerative approach to clustering similar to that of d2_cluster and WCD. However, instead of using d^2, PaCE uses pairwise alignment to determine whether two ESTs should be clustered together. To reduce runtime, PaCE uses the length of a maximal common substring to sort the EST pairs, expecting that EST pairs with longer common substrings should be more likely to pass the pairwise alignment test and be clustered together. PaCE uses a generalized suffix tree (GST) data structure for the maximal common substrings. xsact [13] uses a similar approach, but with a suffix array in place of the suffix tree, and does not use pairwise alignment (the suffix array approach generates the clustering).

CAP3 [14] is a program for full sequence assembly, meaning that the end result is not a set of clusters but a set of consensus sequences derived from the overlaps between ESTs, implicitly

calculating a cluster in the process. Because of the extra work involved in assembly, CAP3 cannot be fairly compared to clustering tools in terms of speed, but the high quality of results makes it very useful for quality comparisons with tools that perform only clustering. To find the overlaps between ESTs, CAP3 uses a BLAST-like [15] approach to filter the potential overlaps among sequences, and then performs alignments to determine the actual overlaps.

TGICL [16] is a program usable for full sequence assembly or individually as a clustering tool. It is essentially a combination of NCBI's megablast [17] (the TGICL authors describe theirs as a slightly modified version) and the aforementioned CAP3 assembly tool [14] in the form of a pipeline. A clustering is initially performed using megablast, and the resulting clusters are fed into CAP3. The tool can also be run in a cluster-only mode using only the megablast component to produce clusters, which is how comparisons were performed with TGICL for this paper.

3. Methods

To solve the problem of clustering gene expressions on a large scale, a program called PEACE (Parallel Environment for Assembly & Clustering of Gene Expressions) [18] was developed in conjunction with the research presented in this thesis. PEACE is designed as an easily modifiable and extensible software system with three main parts: an analyzer (a method to compare sequences and determine overlap, similarity or distance), a cluster maker (a method of building clusters), and, optionally, a chain of one or more heuristics. PEACE also supports parallel processing using the MPI standard. Finally, PEACE contains a fully-featured GUI (graphical user interface) component developed by Dr. Dhananjai Rao. In this section of the thesis, the PEACE software suite will be discussed from the software engineering standpoint, followed by a thorough examination of the algorithms used therein.

3.1 Software design

As mentioned, there are three major components to the PEACE software system: the analyzer, the cluster maker, and the heuristic chain. From the software engineering perspective, PEACE was developed with modularity in mind. That is to say, each type of analyzer, cluster maker and heuristic is a separate object, and any combination of analyzer, cluster maker and heuristic(s) is permissible and functional. This allows for user-specified analysis and clustering strategies and allows PEACE to be not just a tool designed to solve a problem in a single way, as many bioinformatics software tools are designed, but a true framework that can support many different approaches to clustering and even assembly – at the time of this writing, an extension to PEACE is in development that allows the assembly step to take place within the program, instead of forwarding the PEACE clusters to a downstream assembly tool.

PEACE's modularity is provided by the existence of base classes for the major components – ESTAnalyzer, ClusterMaker, and Heuristic – and the use of inheritance to implement specific implementations of those base classes to represent, for example, an ESTAnalyzer that uses the d^2 algorithm. To realize this solution, the program makes use of software design patterns, in

particular the factory design pattern. To select a specific ESTAnalyzer, the program accesses the ESTAnalyzerFactory, which instantiates and returns an instance of the requested analyzer. The same methodology applies to instantiating heuristics and cluster makers. The end result is an elegant software engineering solution that provides substantial abstraction and makes the addition of other algorithms for EST analysis and/or clustering very simple. In this way, PEACE can become a test bed for new ideas and algorithms for gene expression analysis, clustering, and assembly – providing a convenient framework for researchers to implement and test new solutions. An example class diagram, this one for ESTAnalyzer, follows. In the diagram, the different types of analyzers can be seen. FWAnalyzer indicates a frame and word-based analyzer (like the d^2 algorithm, which is based on frames, or windows, and words). TwoPassD2 is the implementation of PEACE's default analyzer, the two-pass d^2 algorithm.

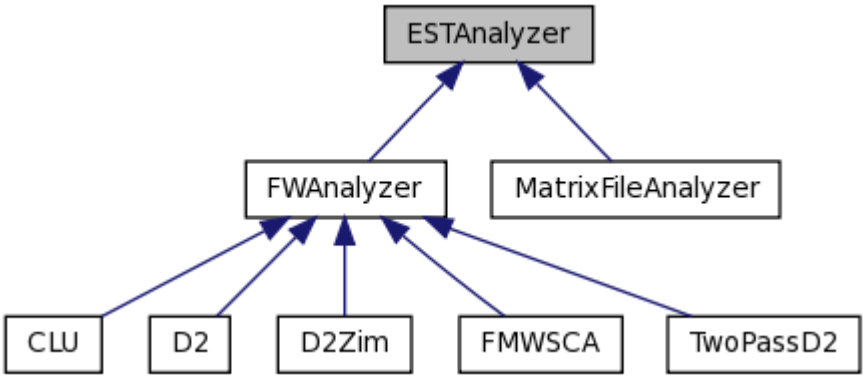

Figure 1: Inheritance diagram for ESTAnalyzer class

The core PEACE software has been written in C++ and is tested and functional on Windows, Mac and Linux operating systems. The GUI was written in Java and also functions on the same three major operating systems. PEACE documentation can be found at the official PEACE website [19].

3.2 Algorithms

The algorithms used with PEACE include a novel d^2-derived algorithm dubbed two-pass d^2 (the "EST analyzer" in the software architectural framework), a minimum spanning tree-based algorithm for clustering (the "cluster maker"), and two filtering heuristics analogous to the heuristics used in the WCD tool: the *uv* and *tv* heuristics, which were discussed in the related works section of this thesis.

The EST analysis algorithm we developed for the PEACE software, dubbed two-pass d^2, makes use of the asymmetric version of the d^2 algorithm [20]. Like symmetric d^2, asymmetric d^2 compares every window on the second sequence, but unlike symmetric d^2, when asymmetric d^2 shifts the window on the first sequence, it shifts by more than one base at a time. Thus asymmetric d^2 runs faster by a factor proportional to the amount skipped each time. It can also be observed that, perhaps surprisingly, asymmetric d^2 returns the same distance as symmetric d^2 more than 50% of the time [20]. The two-pass algorithm therefore works as follows:

1. Run asymmetric d^2 to compare the two sequences.
2. Find the two windows with the shortest distance (best similarity).
3. Initiate a local search centered on these windows, using symmetric d^2, to find the exact distance. This is the d^2 score.

While the d^2 algorithm proved effective in clustering conventional ESTs produced by Sanger sequencing, and the two-pass enhancement provided a substantial improvement in runtime performance, the nature of the algorithm meant that changes needed to be made in order to appropriately cluster reads produced by high throughput sequencing technologies. d^2 assumes a fixed window size, which was typically 100 base pairs in length, and if any sequences are shorter than this window size they will not be clustered. Illumina sequences, for example, are currently 75 base pairs in length, and thus do not work when using the 100 base pair window. In addition, tests on simulated data revealed that the default setting of a 100 base pair window did not work well with sequences of 200 bp in length or less, and tests on real data found that, in certain instances, there could be a wide range of sequence lengths provided (from 100 to 500, for example) and a single static window length could not appropriately cluster these reads.

The adaptive d^2 algorithm was developed to address this problem. The basic principle of adaptive d^2 is that the window length and clustering threshold should not be static, but instead be dynamic and based on the lengths of the two sequences being compared at any given time. This introduces an unavoidable increase in runtime (due to the time needed to calculate and update the lengths) which can be mitigated by caching and pre-calculating the different window lengths needed for comparison between any two reads in the dataset, a technique which was employed in the implementation of the PEACE software. While there was some investigation into an equation that could calculate the ideal window length and clustering threshold on the fly to

accurately compare any two sequences, the marginal gain in quality did not justify the reduced speed and increased complexity. Instead three different groups of sequence lengths were defined and, based on testing, a window length *w* and clustering threshold *T* were assigned to each group, as follows:

Category	Fragment Lengths	*w* (window length)	*T* (clustering threshold)
Short	50-150 bp	50	50
Medium	150-400 bp	75	75
Long	> 400 bp	100	100

Figure 2: Parameters for different sequence lengths

Another unavoidable slowdown is introduced here due to window length: the runtime necessary to perform a d^2 comparison between two sequences is inversely proportional to the window length due to the increase in the number of windows. When a short and a medium-length sequence are compared, the program must use the "short" set of parameters for the comparison. For example, a 50 bp window as compared to a 100 bp window, there will be more total windows in each sequence when using a 50 bp window, and each of those additional windows must be compared with every window in the other sequence (recalling that the d^2 algorithm has an $O(n^2)$ runtime bound). However, as will be demonstrated in the results section of this thesis, the adaptive d^2 enhancement provided high-quality results when applied to simulated 454 and Illumina sequences.

For building the clusters (the "cluster maker"), PEACE makes use of a minimum spanning tree-based algorithm, with ESTs represented as vertexes in the tree and the distances between ESTs represented as edges. The algorithm is essentially derived from Prim's algorithm for minimum spanning trees [21], and involves the following steps:

1. Pick an arbitrary EST to serve as the root.
2. While there is an EST that has not been added to the tree, loop:
    a. Compute the distance between the most recently added EST and all ESTs that have not yet been added to the tree.
    b. Add the resulting edges to a minimum heap-based data structure.

    c. Choose the edge with the minimum distance and add that edge, and the other EST, to the tree.
3. In the resulting tree, edges that exceed a pre-set distance threshold form the delineations between different clusters.

As a departure from the traditional single-linkage/transitive closure algorithms used in other clustering tools, it was believed that the minimum spanning tree-based approach would provide an improvement in clustering quality and/or performance. However, comparison with WCD [10] suggests that there is most likely no change in quality caused by the use of Prim's algorithm for computing a minimum spanning tree. WCD, as has been discussed, utilizes an algorithm called transitive closure to perform single-linkage clustering (the same type of clustering performed by PEACE). Mechanically, the algorithm works as follows:

1. Create a separate cluster for each EST.
2. Calculate the shortest distance between any two clusters.
3. While this distance is less than the clustering threshold:
    a. Merge the two clusters.
    b. Repeat step 2.
4. Return the set of clusters.

In order to perform this algorithm in an efficient manner, WCD makes use of the union-find data structure. As it turns out, this is the same data structure and procedure used in Kruskal's minimum spanning tree algorithm [22]. Thus, the WCD tool is implicitly computing a minimum spanning tree in the process of its clustering.

While the use of the minimum spanning tree may not provide any quality improvement, it is essential to the algorithm implemented in the EAST (EST Assembly using Spanning Trees) tool; PEACE explicitly calculates and is capable of exporting the minimum spanning tree, which is a requirement for that assembly tool to function. Thus, the PEACE and EAST tools, used together, constitute a clustering-assembly pipeline, and scripts have been developed to run the two tools in such a pipeline.

To build the clusters in a parallel environment, the principles of the MST algorithm do not need to be modified, but extra infrastructure is put in place. The processor that is rank 0 (in the MPI

protocol, each processor is assigned a numerical rank starting at 0) serves as the manager processor and all other processors serve as workers. The data set is evenly divided up among all processors such that each processor serves as the "owner" of a range of gene expression fragments. All steps of the MST construction described above can be performed independently by each processor and require no additional input, save for the step of calculating distance between one EST and all other ESTs. This step is done collaboratively, with each processor calculating the distances for all the ESTs it owns, and sending the results back to the manager processor. The manager processor also serves as a worker and has its own set of ESTs to calculate, so it is not waiting for results to come in but actively taking part in the computation. In this way, we parallelize only the critical and time-consuming step of the MST construction, thereby making this a conceptually simple modification for PEACE to run in a parallel environment.

In addition to the analyzer, cluster maker, and heuristics, PEACE introduces a filtering function to eliminate undersized and low-complexity sequences. The filters are implemented in the same way as the other components from the software engineering standpoint, meaning that they are easy to plug in, modify and select. At present PEACE supports two filters. The first is a simple filter that filters out all sequences shorter than 50 base pairs in length, which is the lower bound for the PEACE algorithms to successfully function. Normally, sequencing technologies would not produce reads of this length, but upstream processing may have trimmed off low-quality portions at the beginning and end of the read to reduce its length, and as the length decreases it becomes less and less likely that the read has an information content of any significance or that it can be successfully clustered and/or aligned to other sequences. The second filter is one that can automatically filter out reads with large portions of low-complexity regions, such as repeats. The algorithm used in this filter is very simple and as such it only looks for long single-base repeats or stutters; therefore, it is not particularly useful and is better replaced by an upstream pre-processing tool, but it is present as a proof of concept and demonstrates what the filter functionality is capable of doing.

Putting everything together, the overall workflow of the core PEACE software suite and its component algorithms works as follows:

1. The program is initialized, reads in the appropriate parameters and initializes the various components (analyzer, cluster maker, heuristics, filters).
2. The sequences are read in from a file.
3. Filters, if used, are applied to the sequences, removing reads that are too short or low-complexity.
4. The minimum spanning tree-based clustering algorithm is initialized.
5. The clustering algorithm calls the heuristics and d^2-based distance algorithm to perform comparisons as follows:
    a. The *uv* heuristic is run on the pair of sequences being compared.
    b. If the pair of sequences passes the *uv* heuristic, the *tv* heuristic is run.
    c. If the pair of sequences passes the *tv* heuristic, the adaptive two-pass d^2 algorithm is run to compute the exact distance. This is the most time-consuming step of the whole clustering algorithm.
6. The clustering algorithm chooses the edge with the minimum distance, and continues until the entire minimum spanning tree is produced.
7. The clusters are output to a file, as well as the minimum spanning tree, if desired. The PEACE GUI can be used to provide a much more elegant and understandable interface for analysis and viewing of clustering results. Finally, the clustering results can be sent downstream to the EAST tool for assembly.

4. Results

Several different tests were conducted to properly validate PEACE and the methods used therein as an effective method of clustering gene expression data. First, the tool was tested on simulated data. Simulated data plays an important role in the validation of a computational tool, as with simulated data we can be 100% confident in the "correct answer," and thus can accurately perform a qualitative evaluation of our results. Thus, testing on simulated data is a key first step in validation. Naturally, the next step is to test on real data, and in this thesis tests were conducted on conventional Sanger-sequenced ESTs as well as high throughput data generated by 454 sequencing, from various sources. This section will discuss the construction of a testing

pipeline, the methods used for evaluation of test results, the tests conducted both on simulated and real data, and the interpretation of the test results.

4.1 Testing pipeline

In order to carry out large-scale testing of PEACE on simulated and real data, as well as computational testing on a high-performance computing cluster, a testing pipeline was developed to automate the sequence of simulated EST generation, clustering of the simulated set, and analysis of the results. To generate simulated Sanger-sequenced ESTs, the ESTSim software was used [23]. ESTSim was chosen because of its use for testing and validation of the WCD software – since PEACE was to be compared with WCD this was a necessary step. To generate simulated 454 and Illumina sequences, the software MetaSim [24] was chosen. MetaSim provides specific error models for different types of sequencing (the 454 error model is included with the software, and there are downloadable error models for Illumina reads and others via the MetaSim website [25]) which made it a natural choice for simulated testing on the different high throughput sequencing methods. Perfunctory tests were also conducted using MetaSim's error model for Sanger EST – these showed no noticeable difference in results between ESTSim and MetaSim when PEACE was applied to the simulated Sanger ESTs.

The pipeline itself was written in Python for ease of use and modification. It consists of a primary script (pipeline.py) and several "helper" scripts to perform various roles. The scripts are included with the PEACE source code distribution under the "scripts" directory, and a detailed usage manual is also included in said directory.

The pipeline works as follows for simulated data:

1. Input a set of gene transcripts.
2. Run ESTSim or MetaSim, as appropriate, to generate the simulated gene expression fragments produced by the appropriate sequencing method.
3. Randomly shuffle the resulting fragments – this is to ensure the clustering algorithm works the same regardless of the order in which fragments are input, as it should in theory.
4. Run the PEACE tool on the gene expression fragments, obtaining a set of clusters as output.

5. Calculate the appropriate quality scores/metrics based on the clusters output from PEACE.

The pipeline allows multiple tools (e.g. WCD and PEACE) to be run in parallel on the same set of data, thereby permitting a fair one-to-one comparison on the exact same set and ordering of simulated gene expressions which is necessary for accurate testing. In this case the pipeline will calculate quality metrics and provide output for both tools. The pipeline also keeps track of the runtime to facilitate the computational comparisons.

When tests are performed on real data, the pipeline works in the same fashion, except that steps 1 and 2 are omitted; instead, the pipeline receives as input a set of real gene expression fragments and continues from there.

4.2 Evaluation methods

The nature of simulated data, wherein the exact answer is known by virtue of knowing the gene expressions that produced our simulated fragments, lends itself naturally to the use of a numerical scoring method for quality assessment. In the case of PEACE results, several different quality metrics were calculated.

First and foremost, we look at sensitivity. Sensitivity (se) is the fraction of sequences coming from the same gene transcript that were correctly identified as being so [10]. It works as follows:

Take all pairs of fragments in the data set (a number of fragment pairs equal to n C 2, where n is the total number of fragments).

A *true positive* (tp) is defined as a fragment pair such that both fragments come from the **same** gene transcript and were placed in the **same** cluster.

A *false negative* (fn) is defined as a fragment pair such that both fragments come from the **same** gene transcript but were placed in **different** clusters.

Then, calculate sensitivity by the following formula:

$$se = tp / (tp + fn)$$

Based on this definition we can observe several characteristics of the sensitivity score that point to its value and meaning in result evaluation. We note that sensitivity penalizes for false negatives, or fragments that were put in a different cluster from other fragments belonging to the same gene transcript. When the number of false negatives increases, the denominator increases and the resulting sensitivity score goes down. We can also see that sensitivity will range from 0 (no results) to 1 (perfect results). Finally we notice that there is no penalty for false positives. This means that, if all the fragments in the data set were placed into one cluster (including fragments from many different gene transcripts), the sensitivity score would be 1 – even though such a clustering would be as meaningless as a clustering that placed every fragment in its own singleton cluster and achieved a sensitivity of 0.

We can distill the points above into two meaningful conclusions. Sensitivity is important because when two fragments are incorrectly separated at the clustering stage (a false negative) there is no good way for downstream processes such as assembly to "recover" the proper connection between those fragments without simply scrapping the clustering information. However, because it ignores a different type of error (the false positive), sensitivity cannot be used on its own as the sole scoring measure for evaluating simulated results. Thus, we use an additional scoring metric, the Jaccard Index (JI) [10]. The JI introduces an element of specificity into the formula for sensitivity by adding false positives into the equation.

A *false positive* (fp) is defined as a fragment pair such that both fragments come from **different** gene transcripts but were placed in the **same** cluster.

Calculate the Jaccard Index by the following formula: $JI = tp / (tp + fn + fp)$

The quality measures of Type 1 and Type 2 error assess quality at the level of the genes and clusters as opposed to individual fragments [26]. They provide a different perspective on the same types of errors (mistakenly separating or joining fragments; i.e. false positives and false negatives). Type 1 error is the fraction of genes that were separated into two or more clusters; in other words, the number of "split" genes divided by the total number of genes. Correspondingly, Type 2 error is the fraction of clusters containing two or more genes. These error rates provide perhaps a slightly more intuitive look at the quality results.

A final quality measure is the number of singletons present in the clustering output. A singleton is defined as a cluster containing only one gene fragment. These results can be meaningful, but the vast majority of them are due to inaccurate clustering (i.e. false negatives). The number of singletons can be measured for real as well as simulated data and thus is part of the statistics output for both types of test data.

When it comes to testing clustering tools on real data, it is impossible to use the quality measures discussed above (with the exception of the number of singletons). Thus, it is necessary to test on benchmark data sets where there is a reference clustering available that has been validated by some outside source. In cases where a reference clustering is unavailable, testing the tool on the dataset is not necessarily unimportant; such a dataset can still be useful for the purpose of computational testing (i.e. runtime and varying the number of processors), for generalized comparisons with other tools, and for analysis of the results' feasibility and usefulness in downstream applications, e.g. assembly.

4.3 Test results

Testing began with a set of tests on simulated data based on the tests conducted for the WCD software tool. These tests used a set of 100 zebrafish genes as the base genes from which simulated ESTs were generated, using the ESTSim software. The zebrafish genes come from the WCD paper and related data [10]. For the initial tests, the number of ESTs was fixed and the single-base error rate was varied. The results presented for each error rate were averaged over 30 trials and for each trial, the tools PEACE, WCD and CAP3 were run on the same simulated data set.

Single-base error rate means the probability that one base will be changed to a different base (A, C, G, or T) to simulate the effect of base read errors in the sequencing. ESTSim models these errors as well as single-base insertions and deletions, N bases (meaning the sequencer could not determine which base was present at that location and placed an N character there instead), and also models errors due to polymerase decay and primer interference (which essentially result in a higher rate of error near the beginning and end of the EST, in accordance with what occurs in real Sanger sequencing). All settings, other than the single-base read error rate, were identical to

settings used in the testing of WCD. Figure 3 shows the sensitivity results (see the Appendix for the other quality measures).

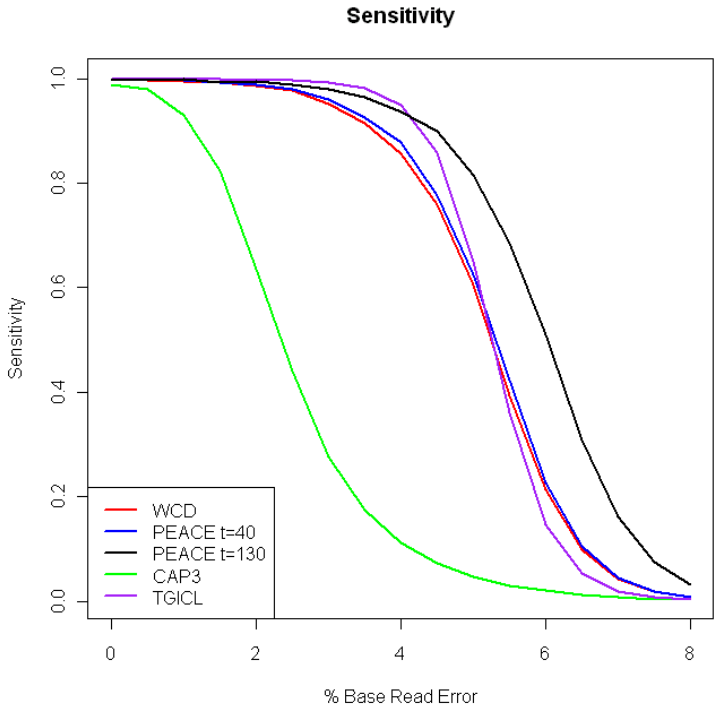

Figure 3: Sensitivity comparison between different tools

From these results, we observe that PEACE achieves results comparable to WCD's in sensitivity, and outperforms CAP3. PEACE performs worse than TGICL at lower error rates but better at higher error rates. In testing, it was also found that by increasing the clustering threshold (the maximum d^2 score between two sequences for which those sequences would still be placed in the same cluster) PEACE achieved an even higher level of sensitivity on the simulated EST data sets with a comparatively small increase in the incidence of false positives. However, it was found that the higher threshold, while effective in theory, caused problems in practice: using a higher threshold greatly increased the incidence of false positives in real data sets. This is because those false positives were due to other errors and phenomena that are not modeled by the ESTSim tool, a topic discussed further in section 4.4 along with the "supercluster" phenomenon. The difference between lower and higher clustering thresholds can be seen in Figure 3 with the PEACE t=40 (lower threshold) and PEACE t=130 (higher threshold) entries.

Along with the tests on different error rates, we tested the ability of PEACE and the other tools to distinguish between duplicated genes that were allowed to diverge at a certain rate. Mechanically, this was done by copying the gene and changing each base with a probability equal to the rate specified, then deriving simulated ESTs from both genes and running PEACE and the other tools on the combined file of ESTs. These results are seen in Figure 4 (see the Appendix for the TGICL data).

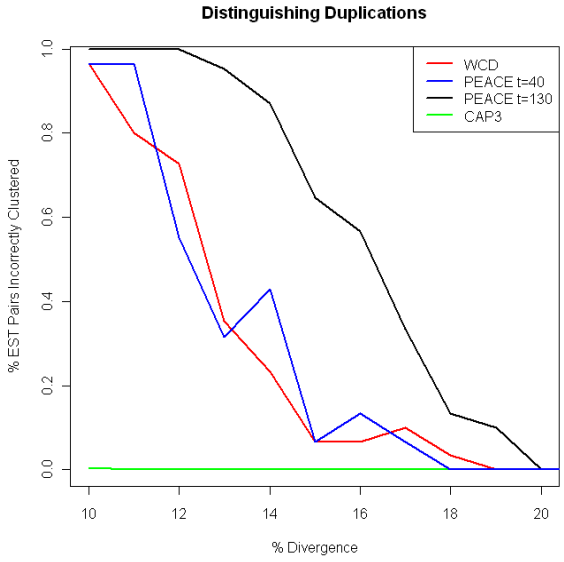

Figure 4: Distinguishing duplications

The duplication results show that PEACE is again comparable to WCD. CAP3 performs much better than both of the d^2-based clustering tools when it comes to distinguishing duplications. Testing on TGICL (seen in the Appendix) showed a substantial gap between PEACE and TGICL in their ability to distinguish duplications, with PEACE proving superior.

As discussed in section 4.1, we employed the tool MetaSim to test on simulated short read data, using both a 454 and an Illumina error model. It should be noted that the supplied Illumina error model used 62 bp reads; as of this writing, typical Illumina reads are slightly longer, but the available error models were 62 and 80 bp respectively, and we wanted to test the ability of PEACE and the other tools to handle the shortest reads available. The results are summarized in the following table.

Tool	Sensitivity	Jaccard	Type 1	Type 2	Number of	Number of	Runtime

			Index	Error	Error	Clusters	Singletons	
MetaSim Simulation 454	PEACE	0.849	0.818	0.269	0.022	138	4	129s
	WCD[1]	0.459	0.448	0.946	0.003	599	235	76s
	CAP3	-	-	-	-	-	-	-
	TGICL	0.205	0.201	1.000	0.001	5352	4470	68s
Illumina (62 bp reads)	PEACE	0.383	0.342	0.978	0.019	680	233	3463s
	WCD	-	-	-	-	-	-	-
	CAP3	-	-	-	-	-	-	-
	TGICL	0.128	0.125	1.000	0.006	2305	770	373s

As the above data shows, CAP3 did not run on the data from either set. WCD was able to cluster the 454 reads using alternative parameters (see footnote) but was unable to cluster the Illumina reads. TGICL ran on both data sets, but clearly PEACE outperformed all other tools by a wide margin, having been developed with the requirement of handling this type of data and not only conventional Sanger-sequenced ESTs.

Moving on to real data, we tested using several benchmark sets, again as discussed in section 4.1. The benchmarks for which we had high-quality reference clusterings were the EasyCluster [27] and WCD [10] benchmarks, from human and Arabidopsis respectively, and the quality results for these benchmarks follow.

Benchmark	Tool	Sensitivity	Jaccard Index	Type 1 Error	Type 2 Error	Number of Clusters	Number of Singletons	Runtime
EasyCluster Human (111 genes)	PEACE	0.998	0.672	0.153	0.042	118	21	293s
	WCD	0.998	0.672	0.144	0.044	113	16	804s
	CAP3	0.657	0.643	1.000	0.001	2269	1827	N/A
	TGICL	0.998	0.949	0.568	0.018	221	86	278s
WCD A076941 (13240 genes)	PEACE	0.932	0.475	0.351	0.027	18825	8951	1166s
	WCD	0.933	0.476	0.350	0.027	18787	8553	966s
	CAP3	0.826	0.802	0.486	0.014	25042	14916	N/A
	TGICL	0.939	0.209	0.401	0.020	20248	1065	425s

[1] Used different parameters for WCD as suggested by the authors: "-H 1 –T 160"

Here we can see that for the EasyCluster benchmark [27], PEACE, WCD and TGICL achieved equivalent sensitivity, with PEACE and WCD being very close in all other categories save runtime, where PEACE wins. TGICL had a higher Jaccard index but also a higher rate of Type 1 error than PEACE and WCD. The increase in Type 1 error is due to the fact that TGICL had about 100 more clusters than did PEACE and WCD, meaning it incorrectly separated more gene transcripts than the other two clustering tools. On the A076941 dataset [10], we see a similar pattern, but here it is less pronounced.

The next table shows runtime results of PEACE v. WCD on real data benchmarks. PEACE outperforms WCD serially but is worse on the parallel set – this is discussed in section 4.4.

Benchmark	Number of Sequences	Number of Bases	PEACE Runtime (s)	WCD Runtime (s)
EasyCluster Human	**17333**	**11.7 Mb**	**293s**	**804s**
Cotton	**29995**	**17.1 Mb**	**222s**	**339s**
Ricinus	**57690**	**40.9 Mb**	**1344s**	**2356s**
A076941	**76941**	**32.8 Mb**	**1166s**	**966s**
A686903[2]	**686903**	**294.7 Mb**	**9449s**	**2481s**

To continue with the computational analysis of the clustering tools, runtimes were tested on various single-processor and multi-processor scenarios using simulated data generated by ESTSim. For this part of the testing, only WCD, PEACE, and TGICL were tested. CAP3 falls behind the other tools in runtime because it always performs full sequence assembly, and thus a clustering tool beating CAP3 in runtime is a trivial exercise and not important to the testing. The single processor runtime graph shows WCD and PEACE keeping pace and TGICL bettering the other tools as the size of the EST set increases.

[2] Run over 30 processors; the other runs were sequential.

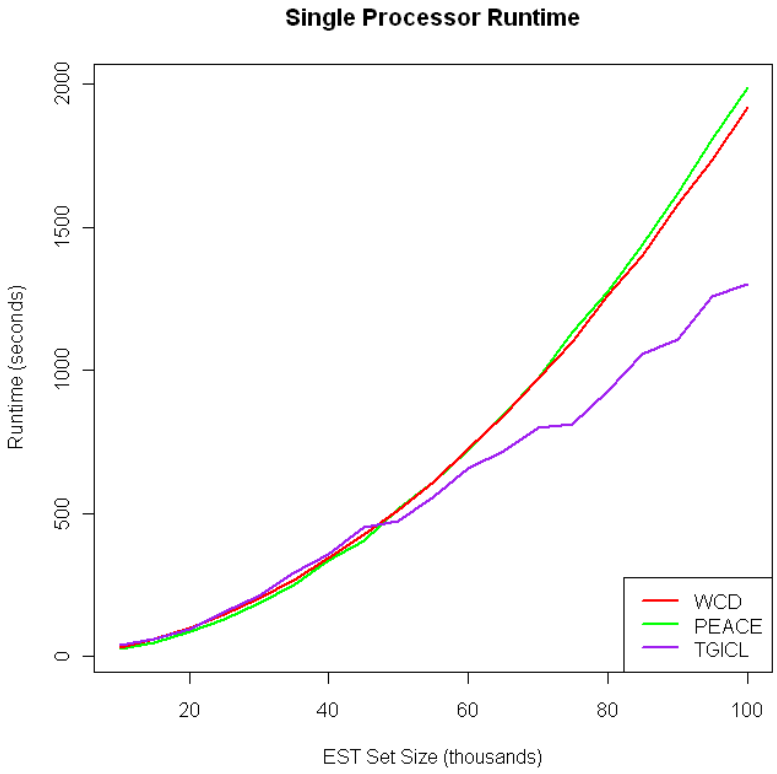

Figure 5: Single processor runtime comparison

In the testing of WCD and PEACE on multiple processors, it was observed that WCD tends to catch up with PEACE as the number of processors increases – the mechanical reasons for this are discussed in section 4.4. Instead of performing a simple test of keeping the number of ESTs fixed and increasing the number of processors linearly, we attempted to find the ratio of EST set size increase per processor that caused the two tools to keep pace. This can be seen in Figure 6.

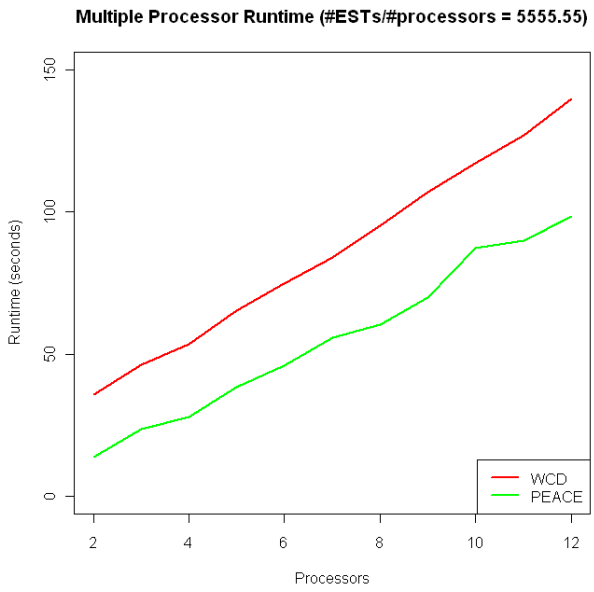

Figure 6: Multiple processor testing

Finally, we tested the memory usage of PEACE, WCD, and CAP3. As would be expected for full sequence assembly, CAP3 uses much more memory than the other tools. PEACE uses somewhat more than WCD but both manage to be comparatively low in memory consumption.

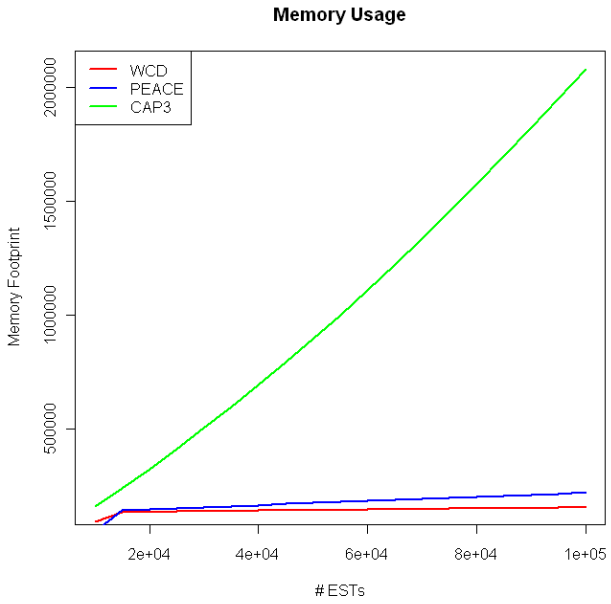

Figure 7: Memory usage

4.4 Interpretation

The application of PEACE to several non-benchmark real data sets does raise questions regarding the utility of the method being used and/or the characteristics of the gene expressions involved. In one case, PEACE was applied to a set of approximately 220,000 Sanger ESTs sequenced from the *Chlamydomonas reinhardtii* genome. The chief observation in this case was that PEACE put an overwhelming portion of the ESTs – 102,135 of them, or 46.39% of the total 220,181 ESTs – into the same cluster. The WCD tool produced very similar results, but CAP3, the assembler, did not, correctly separating most of the fragments into different contigs (producing a Jaccard Index of 0.710 as opposed to the 0.005 achieved by WCD and PEACE on the same data). An analysis of gene expression fragments generated by 454 pyrosequencing from the same genome showed similar results, as did tests on pine data (*Pinus taeda*) of similar characteristics. These results, and the persistent indications of what we term as "superclusters," raise several relevant questions.

The problem of superclusters, or overly large clusters, does seem to be a consequence of the agglomerative single-linkage approach to clustering implemented in WCD and PEACE and not necessarily unique to PEACE as a tool [28]. The TGICL package in fact includes a sizeable section about "Dealing with large clusters" in the documentation bundled with the software [16]. In that documentation it is noted that there is a possibility the cluster is only coming from one highly expressed gene with very deep coverage. However, in the examples mentioned above, this is clearly not the case – Sanger sequencing does not produce such a depth of sequencing and in the case of the *Chlamydomonas* data, other tools were able to characterize the different ESTs in the supercluster as being derived from different genes and, in so doing, break up the supercluster. Far more likely (and common) is the case that, as they note, several (or potentially many) genes are joined in the same cluster due to chimeric ESTs, or repeats and contaminations in the ESTs. Because the clustering algorithm requires only a single link between two clusters to join them together (for the minimum spanning tree this means only one edge) it is possible to form a long chain of fragments deriving from different gene expressions by finding the places where they overlap, which is exactly what the d^2 algorithm does.

How, then, to address the problem? The first part of the answer is cleaning – it is essential to ensure the data coming in is clean, meaning that vector sequences have been removed; repeats

have been masked out, etc. This illustrates the importance of data pre-processing in any bioinformatics pipeline. However, repeat masking proved to have little effect on the supercluster problem in the *Chlamydomonas* dataset. Detailed analysis of the problem ESTs in the dataset in conjunction with the GMAP [29] tool did show that in most cases, clusters were joined together due to what appeared to be chimeric ESTs or transcripts – where a fraction of the EST mapped to one gene and the other part mapped to a different gene (although it is possible these could be GMAP errors). This is difficult enough to deal with on its own, and the presence of duplicated regions or duplicate genes in the genome (another biological phenomenon that could easily lead to superclusters) would only complicate matters further. Essentially, we have here a computational genomics problem that is beyond the scope of this work.

To remain within the context of this thesis, the most relevant question is: what impact does the supercluster phenomenon have on the usefulness of PEACE as a tool? When used as part of a pipeline that includes an assembly tool downstream of PEACE's clustering, the supercluster at least does not have a negative impact, because the assembler (e.g. CAP3) should be able to break up the superclusters. However, it is true that when one cluster takes up nearly 50% of the data set, clustering as a pre-assembly step loses much of its usefulness. Thus, PEACE is not as effective as it could be with the superclusters gone, but as has been established, their existence is inherently tied to the method of agglomerative single-linkage clustering which forms the core of the algorithm.

Another issue revealed during testing concerns the scalability of PEACE in multi-processor mode. When compared to the WCD tool, PEACE is able to keep pace as long as the number of ESTs increases proportionally with the number of processors (as seen in the test results section). However, if the number of processors is increased and the number of ESTs is fixed, WCD experiences a greater relative gain in performance and eventually overtakes PEACE in overall performance. In effect, when comparing to WCD on the ratio-based tests, the improved efficiency of PEACE's algorithms is offsetting the reduced efficiency of PEACE's approach to parallelization.

There are two factors that account for PEACE's reduced scalability and could, if addressed, result in performance gains when run in parallel mode. The first is the duties of the manager process. The parallel implementation of WCD uses a manager processor that does no other tasks

beyond sending and receiving information from the worker processes. This can be observed by testing WCD's performance using two processes, compared to only one process; there is no performance gain for using two processes. PEACE, on the other hand, uses a manager that has the same computational responsibilities as a worker process in addition to its managerial component; in effect, it is a "manager-worker" process. This results in greater performance for smaller numbers of processes, but as the number of processes increases past 10 or 12, the cost of communication offsets the benefit of having the manager perform computations, and PEACE gains less and less runtime improvement from each additional processor.

The other factor is more elusive, but it results from the possibility of the computational work being unevenly distributed among the PEACE processes. Since each process is assigned a block of ESTs of equal size, and we know nothing about the characteristics of those ESTs, it is exceedingly likely that some processes will end up doing more work than others. This comes from two sources. One source is the fact that with each EST added to the minimum spanning tree, we no longer need to calculate the distance to that EST, therefore decrementing by 1 the number of calculations that must be performed by the EST's owner process. The other source is that we do not know how many heavy weight d^2 comparisons must be performed among all the comparisons to be done – more than 90% of comparisons will not pass the heuristics and thus be much less costly, and depending on how the remaining comparisons are distributed among the processes, some processes will do more work than others. These factors will combine to result in wasted CPU time for some of the worker processes. Addressing this source of parallel slowdown would be difficult compared to the "manager-worker"-induced slowdown. It would likely involve a system of dynamic load balancing, including diagnostic communications between the manager and workers to determine where and how the reallocation of owned ESTs needs to take place. Overall, it would probably take a complete rewrite of the minimum spanning tree-based cluster maker logic, an effort that the PEACE software infrastructure would support but that has not been undertaken as of this writing.

5. Conclusions

This thesis has presented PEACE, a fully parallelized software system and framework for the clustering and analysis of gene expressions. PEACE employs the d^2 algorithm and established heuristics derived from the WCD tool with novel extensions and additions to deal with the wide variety of gene expression data available, including short read data from modern sequencing technologies such as 454 pyrosequencing and Illumina sequencing by synthesis. PEACE uses a clustering algorithm based on minimum spanning trees (MSTs) and shows significant improvement over the competing tools WCD, TGICL, and CAP3 when applied to next generation sequencing (NGS) data, as well as matching WCD's performance and providing better sensitivity than CAP3 on ESTs generated by conventional Sanger sequencing.

The algorithms and methods used in PEACE have limitations, as this work has established. Chief among those is the vulnerability to "superclusters," which have several possible explanations including poorly processed data, trans-splicing, and/or an abundance of duplicate genes or regions thereof in the gene expression fragments being analyzed. The supercluster phenomenon is one that warrants further investigation. In addition, PEACE as a computer science tool has room for improvement, particularly the efficiency with which it makes use of parallel processing power.

It is hoped that the modularity of the PEACE software system, the free availability of the software and source code, and the research performed in the course of this thesis constitute a solid foundation upon which future work may be built.

Bibliography

1. 454 Life Sciences, a Roche Company. (2010). http://www.454.com/

2. Illumina, Inc. (2010). http://www.illumina.com/

3. Applied Biosystems by Life Technologies. (2010). http://www.appliedbiosystems.com/

4. Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology, 147*(1), 195-197.

5. Torney, D. C., Burkes, C., Davidson, D., & Sirkin, K. M. (1990). Computation of d2: A measure of sequence dissimilarity, computers and DNA, SFI studies in the sciences of complexity. In G. Bell, & T. Marr (Eds.), (). New York, NY: Addison-Wesley.

6. Hide, W., Burke, J., & Davison, D. B. (1994). Biological evaluation of d2, an algorithm for high-performance sequence comparison. *Journal of Computational Biology : A Journal of Computational Molecular Cell Biology, 1*(3), 199-215.

7. Hazelhurst, S. (2004). An efficient implementation of the d2 distance function for EST clustering: Preliminary investigations. *SAICSIT '04: Proceedings of the 2004 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists on IT Research in Developing Countries,* Stellenbosch, Western Cape, South Africa. 229-233.

8. Nagaraj, S. H., Gasser, R. B., & Ranganathan, S. (2007). A hitchhiker's guide to expressed sequence tag (EST) analysis. *Briefings in Bioinformatics, 8*(1), 6-21.

9. Burke, J., Davison, D., & Hide, W. (1999). d2_cluster: A validated method for clustering EST and full-length cDNAsequences. *Genome Research, 9*(11), 1135-1142.

10. Hazelhurst, S., Hide, W., Liptak, Z., Nogueira, R., & Starfield, R. (2008). An overview of the wcd EST clustering tool. *Bioinformatics, 24*(13), 1542-1546.

11. Hazelhurst, S. (2008). Algorithms for clustering expressed sequence tags: The wcd tool. *South African Computer Journal,* (40)

12. Kalyanaraman, A., Aluru, S., Kothari, S., & Brendel, V. (2003). Efficient clustering of large EST data sets on parallel computers. *Nucleic Acids Research, 31*(11), 2963-2974.

13. Malde, K., Coward, E., & Jonassen, I. (2003). Fast sequence clustering using a suffix array algorithm. *Bioinformatics (Oxford, England), 19*(10), 1221-1226.

14. Huang, X., & Madan, A. (1999). CAP3: A DNA sequence assembly program. *Genome Research, 9*(9), 868-877.

15. Altschul, S. F., Gish, W., Miller, W., Myers, E. W., & Lipman, D. J. (1990). Basic local alignment search tool. *Journal of Molecular Biology, 215*(3), 403-410.

16. Pertea, G., Huang, X., Liang, F., Antonescu, V., Sultana, R., Karamycheva, S., et al. (2003). TIGR gene indices clustering tools (TGICL): A software system for fast clustering of large EST datasets. *Bioinformatics (Oxford, England), 19*(5), 651-652.

17. Zhang, Z., Schwartz, S., Wagner, L., & Miller, W. (2000). A greedy algorithm for aligning DNA sequences. *Journal of Computational Biology : A Journal of Computational Molecular Cell Biology, 7*(1-2), 203-214.

18. Rao, D. M., Moler, J. C., Ozden, M., Zhang, Y., Liang, C., & Karro, J. E. (2010). PEACE: Parallel environment for assembly and clustering of gene expression. *Nucleic Acids Research, 38 Suppl*, W737-42.

19. PEACE-Tools. (2010). http://www.peace-tools.org/

20. Zimmermann, J. (2003). *Suitability comparison of string distance measures for EST clustering.* Unpublished Master's, ETH Zurich, Dept. of Computer Science, ETH Zurich, Dept. of Computer Science.

21. Prim, R. C. (1957). Shortest connection networks and some generalizations. *Bell System Technical Journal.*

22. Kruskal, J. B. (1956). On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society, 7*(1), 48-49, 50.

23. Zimmermann, J., Liptak, Z., & Hazelhurst, S. (2004). *A method for evaluating the quality of string dissimilarity measures and clustering algorithms for EST clustering*.

24. Richter, D. C., Ott, F., Auch, A. F., Schmid, R., & Huson, D. H. (2008). MetaSim--A sequencing simulator for genomics and metagenomics. *PLoS ONE, 3*(10), e3373.

25. MetaSim -- algorithms in bioinformatics. (2010). http://www-ab.informatik.uni-tuebingen.de/software/metasim

26. Wang, J. P., Lindsay, B. G., Leebens-Mack, J., Cui, L., Wall, K., Miller, W. C., et al. (2004). EST clustering error evaluation and correction. *Bioinformatics (Oxford, England), 20*(17), 2973-2984.

27. Picardi, E., Mignone, F., & Pesole, G. (2009). EasyCluster: A fast and efficient gene-oriented clustering tool for large-scale transcriptome data. *BMC Bioinformatics, 10 Suppl 6*, S10.

28. Bragg, L. M., & Stone, G. (2009). k-link EST clustering: Evaluating error introduced by chimeric sequences under different degrees of linkage. *Bioinformatics (Oxford, England), 25*(18), 2302-2308.

29. Wu, T. D., & Watanabe, C. K. (2005). GMAP: A genomic mapping and alignment program for mRNA and EST sequences. *Bioinformatics (Oxford, England), 21*(9), 1859-1875.

Appendix

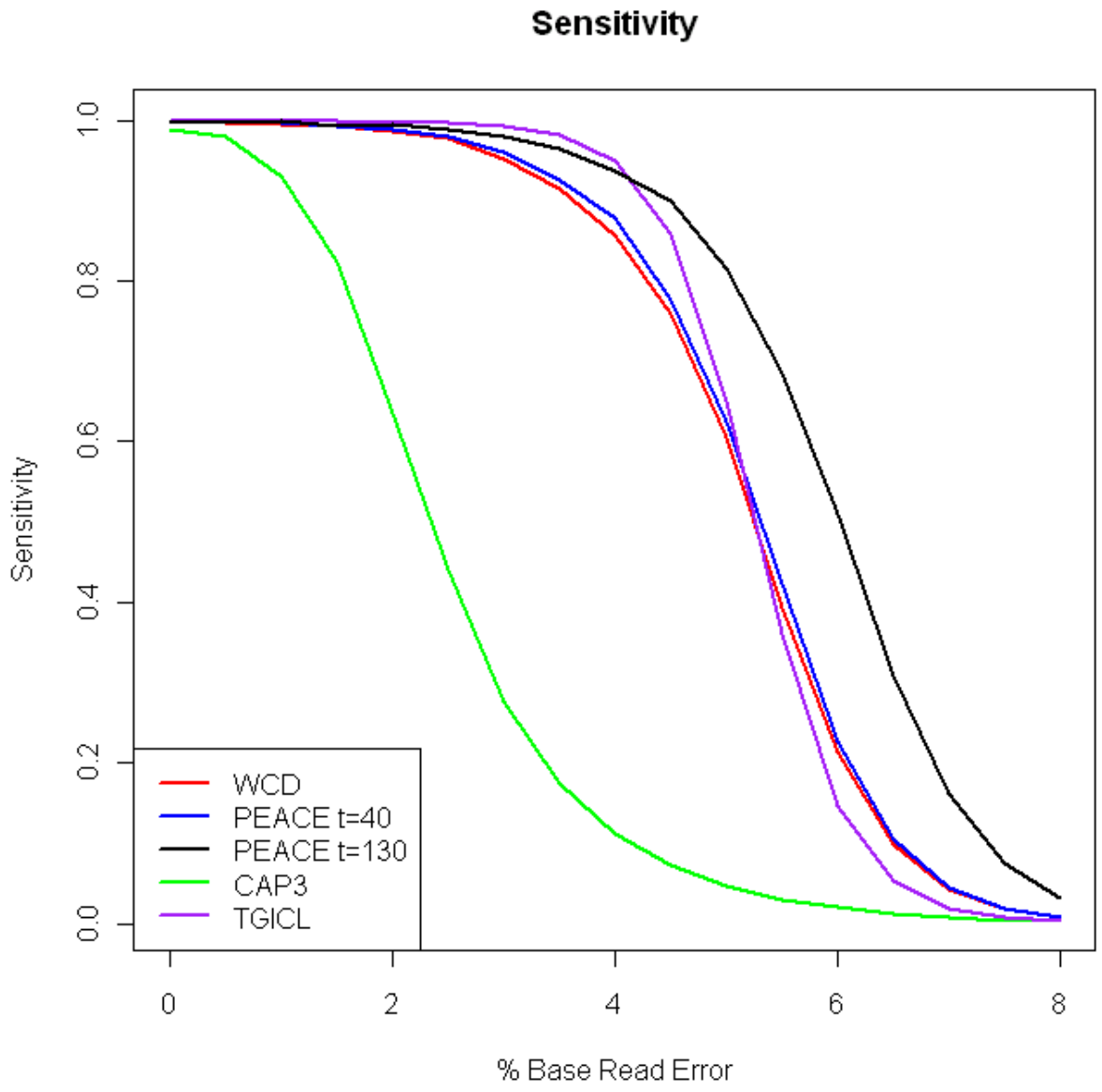

Figure 8: Sensitivity (full-size)

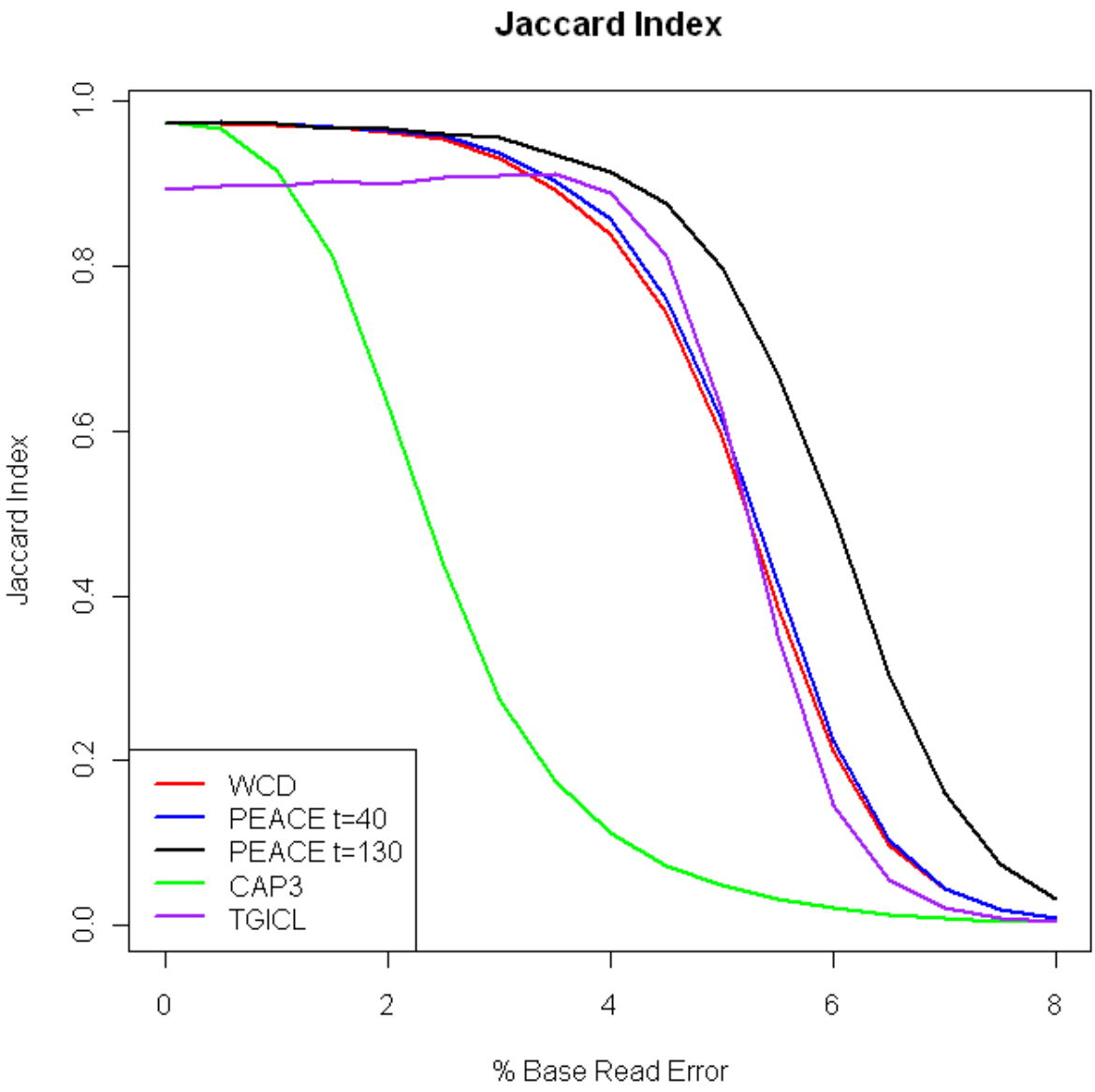

Figure 9: Jaccard Index (full-size)

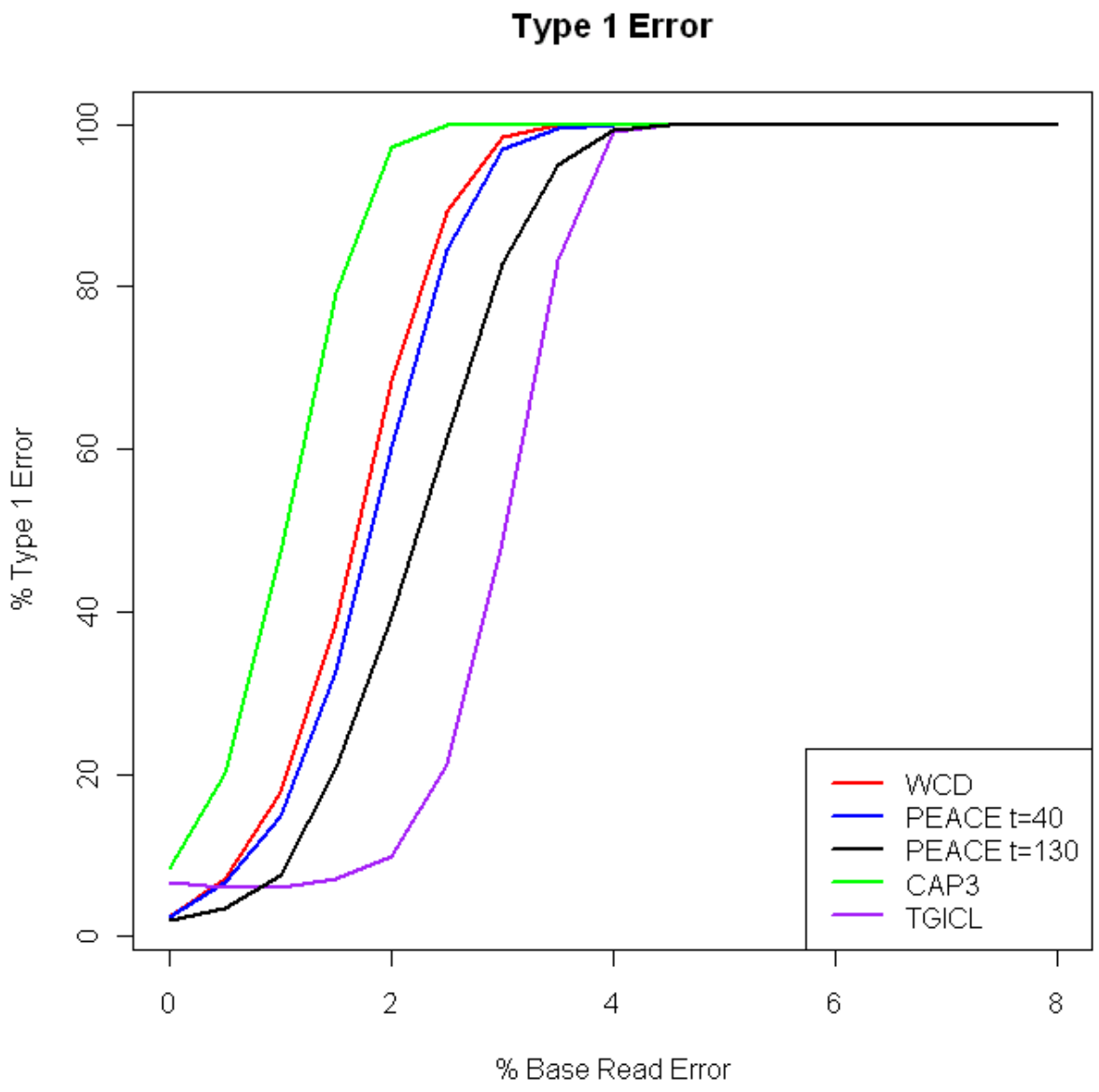

Figure 10: Type 1 Error (full-size)

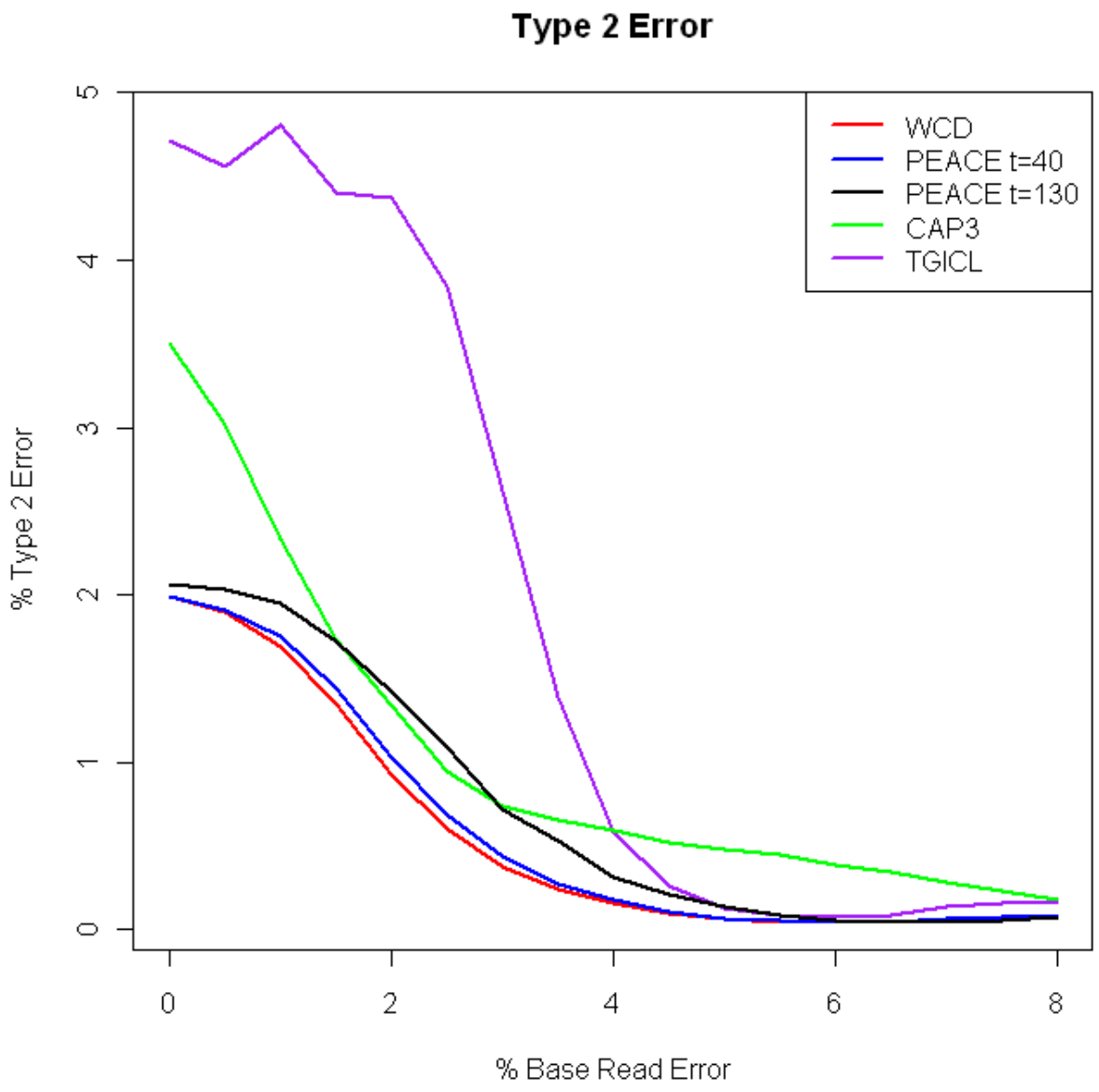

Figure 11: Type 2 Error (full-size)

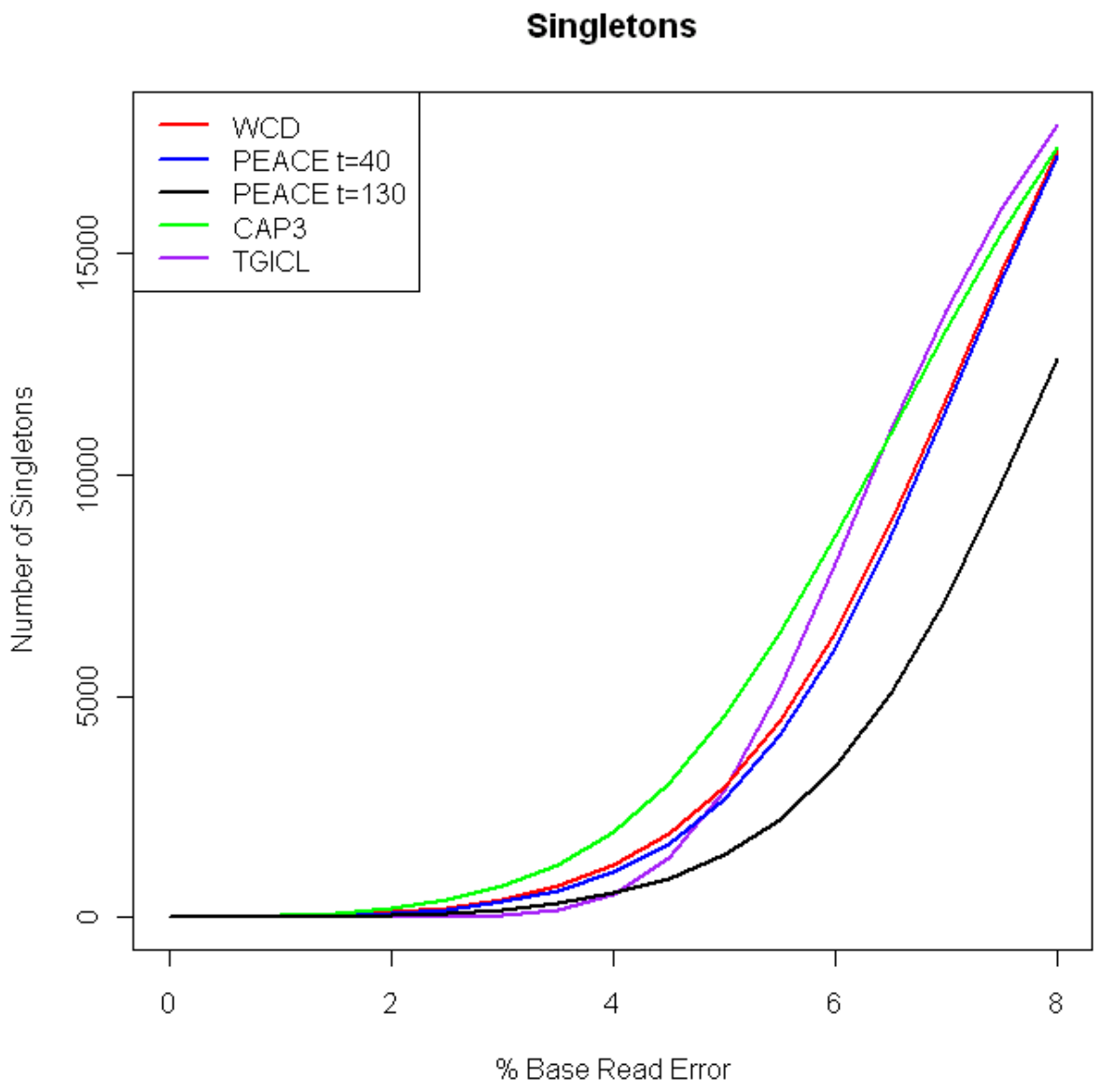

Figure 12: Singletons (full-size)

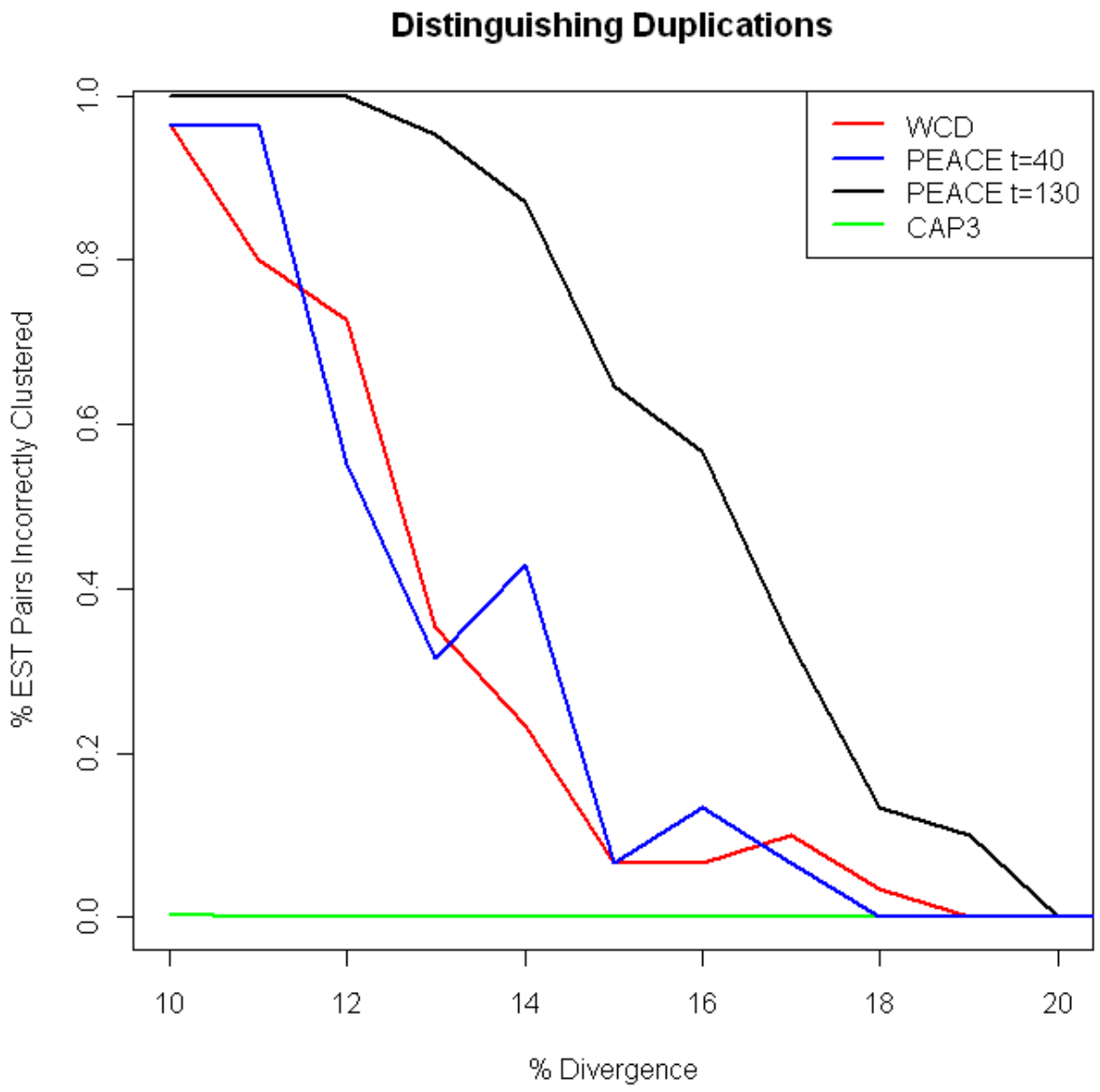

Figure 13: Distinguishing Duplications (full-size)

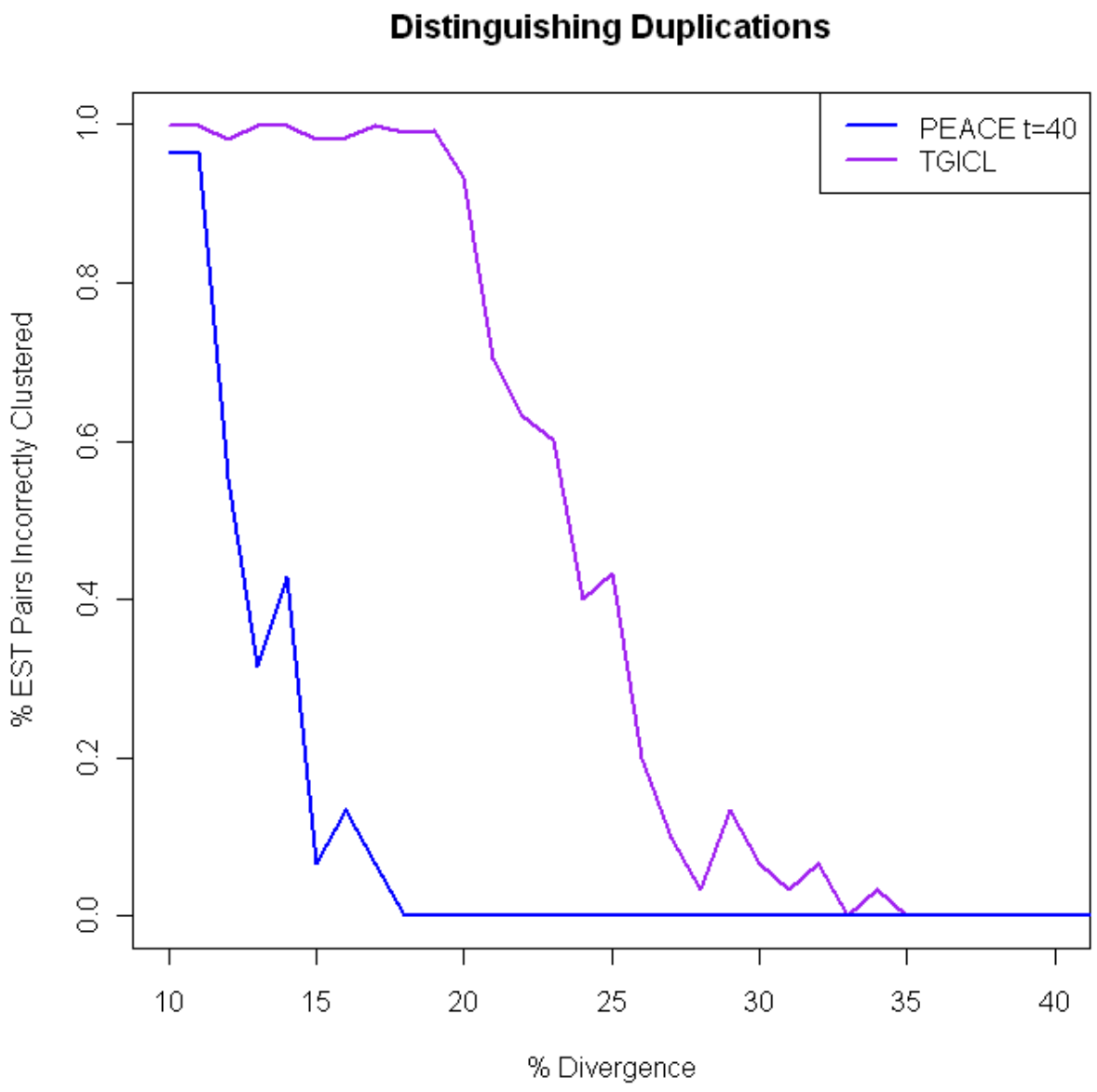

Figure 14 : Distinguishing Duplications, TGICL (full-size)

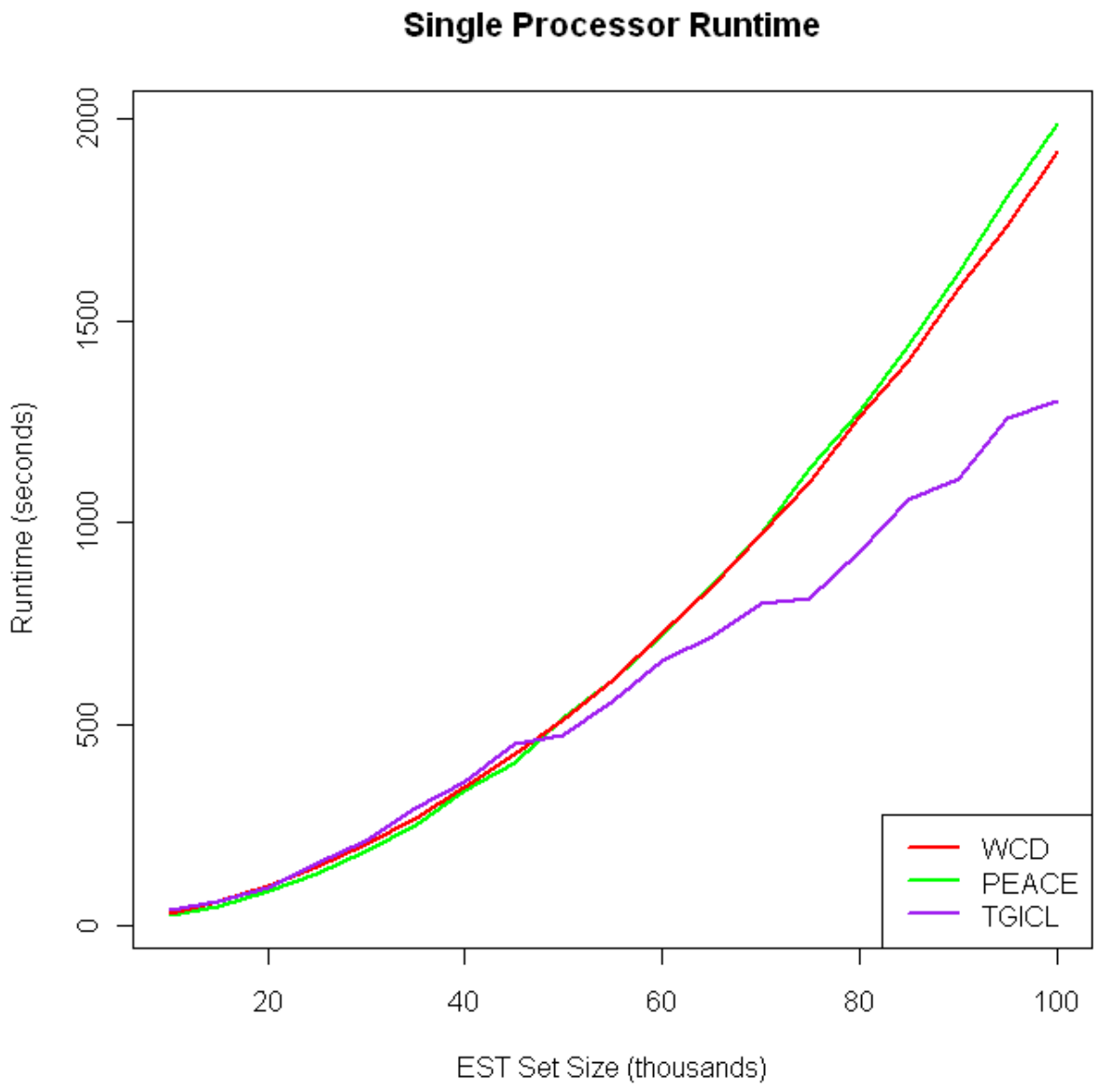

Figure 15: Serial Runtime (full-size)

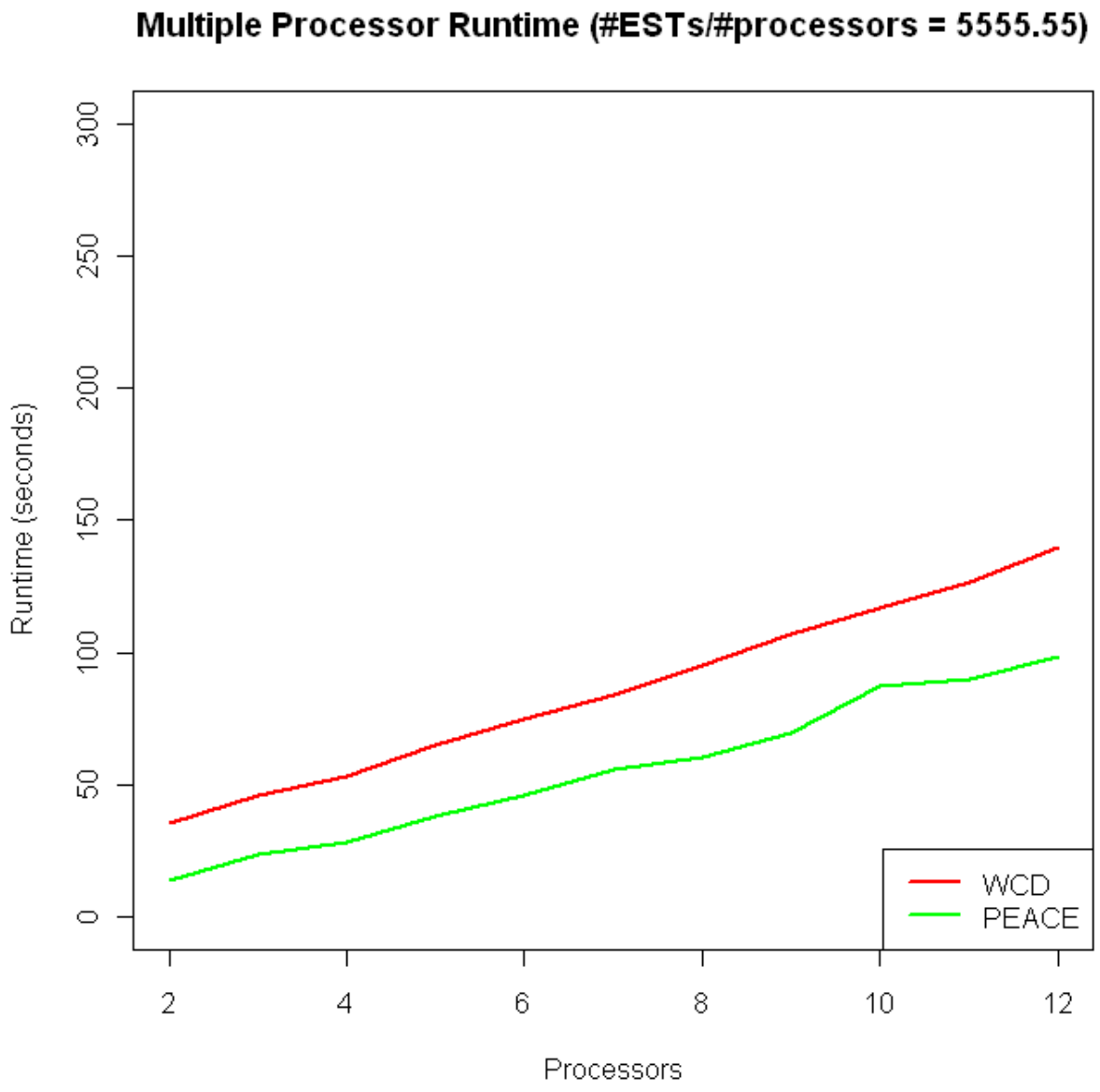

Figure 16: Parallel Runtime, 5555.55 (full-size)

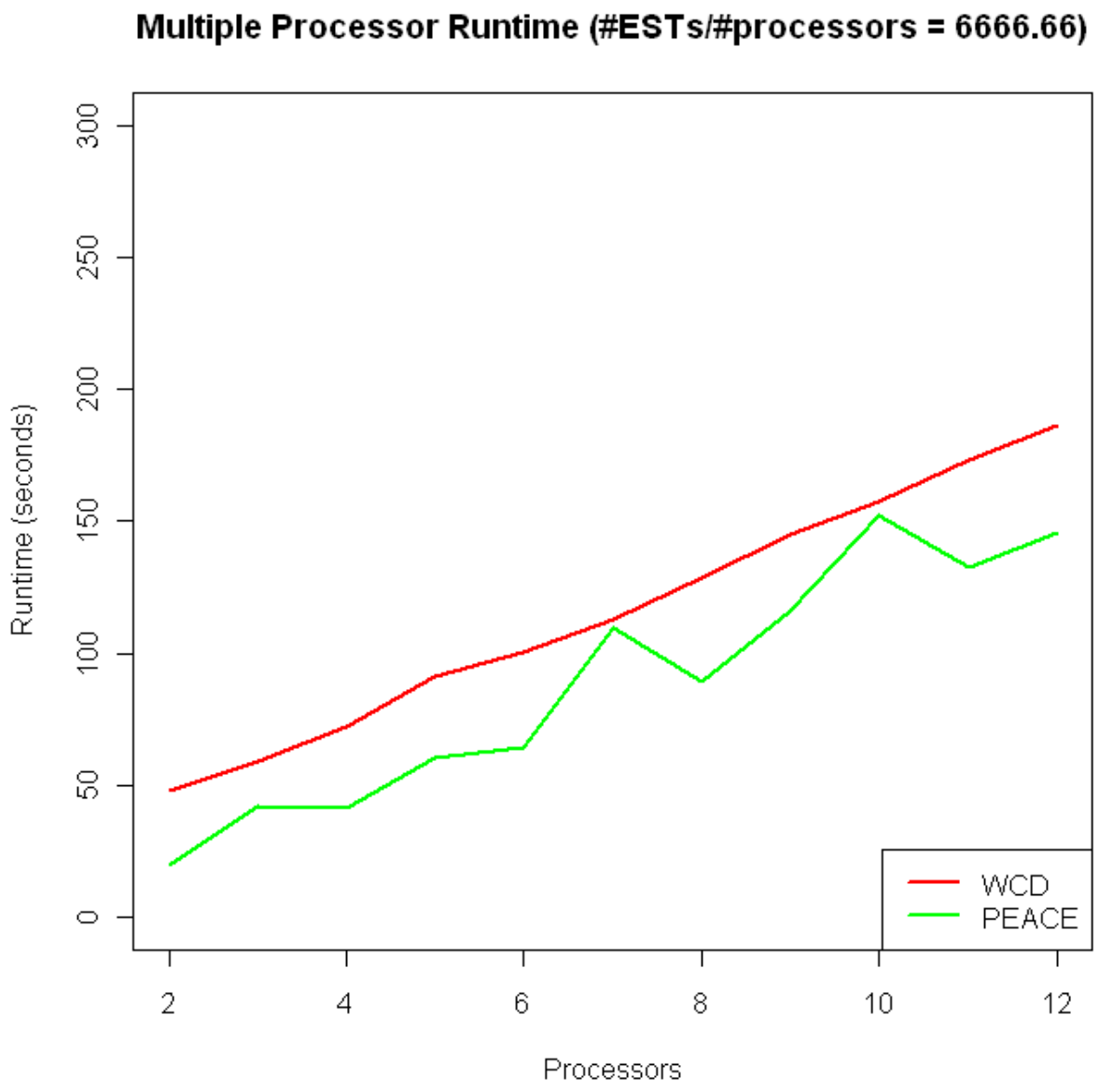

Figure 17: Parallel Runtime, 6666.66 (full-size)

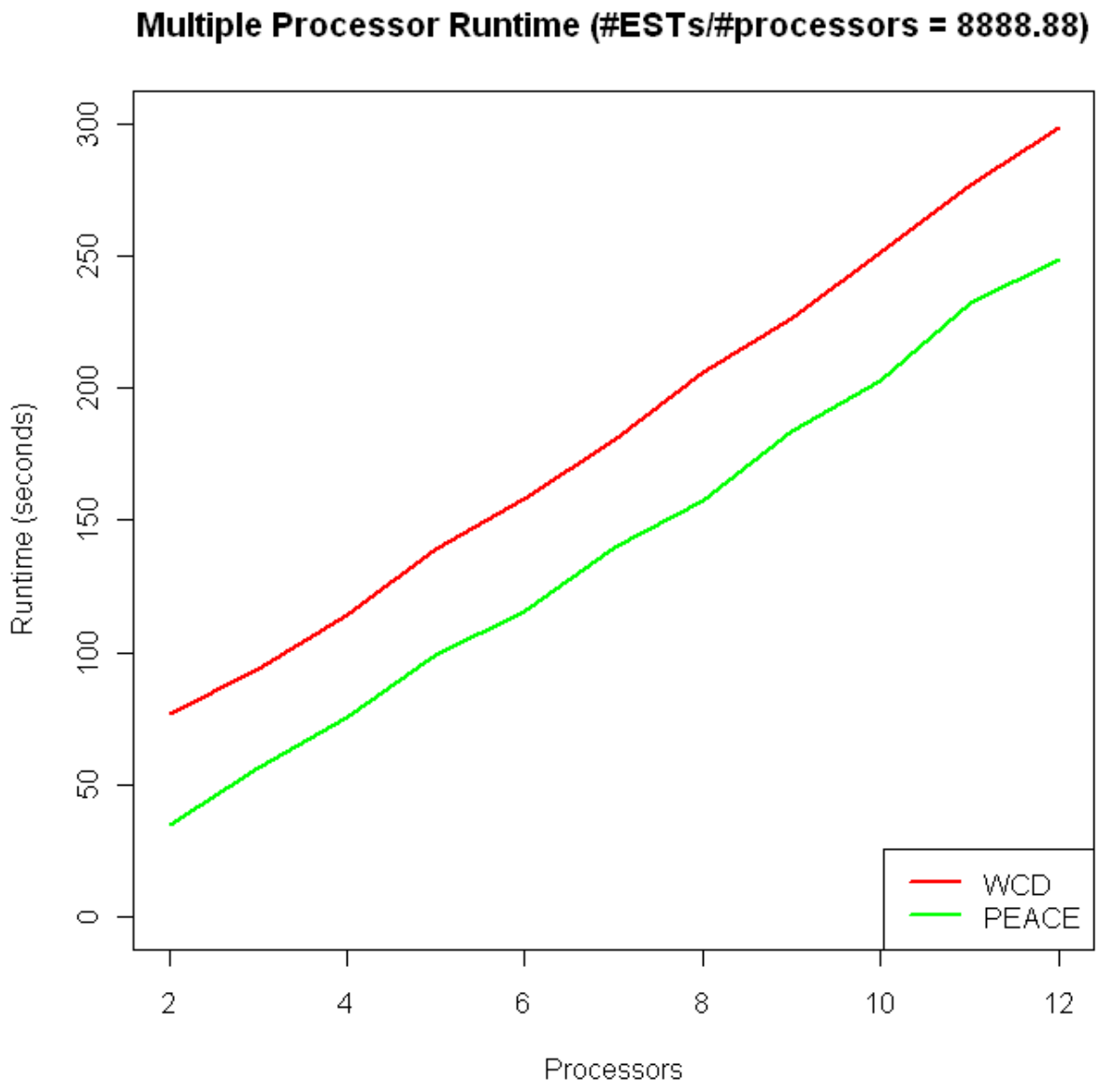

Figure 18: Parallel Runtime, 8888.88 (full-size)