University of California, Santa Cruz

PhD Advancement

Large-scale Solutions for Genomic Analysis

Author:
John Vivian

Advisors:
Dr. David Haussler
Dr. Benedict Paten
Dr. Gunnar Rätsch
Dr. Josh Stuart

A thesis submitted in fulfillment of the requirements for advancement to candidacy

Computational Genomics Lab
Biomolecular Engineering Department

August 20, 2017

Declaration of Authorship

I, John Vivian, declare that this advancement thesis and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this capstone has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

"For me, it is far better to grasp the Universe as it really is than to persist in delusion, however satisfying and reassuring."

Carl Sagan

"Take the risk of thinking for yourself — much more happiness, truth, beauty, and wisdom will come to you that way."

Christopher Hitchens

Acknowledgements

I would like to acknowledge and thank my committee: David Haussler, Benedict Paten, Gunnar Rästch, and Josh Stuart for taking time out of their busy schedules to advise me on my work and provide critical feedback.

I would also like to acknowledge and thank my colleagues: Arjun Rao, Ian Fiddes, Nathan Schafer, Edward Rice, Arthur Rand, and Miten Jain. My lab mates: Jordan Eizenga, Charles Markello, Joel Armstrong, Yoehi Rosen, Molly Zhang, Jacob Pfeil, Adam Novak, and Alden Deran. I'd also like to thank the Computational Genomics Lab, the wonderful Genomics Institute staff, and the countless others at UC Santa Cruz whose positive interactions have inspired me.

Finally, I would like to thank my parents for all their love, support, and encouragement for the endeavors I have chosen to pursue in my life.

Contents

Abstract

As genomic technologies advance and sequencing costs continue to fall, it is necessary to develop methods for managing and processing the ever-increasing size of data, as well as scaling downstream analyses to leverage this data effectively. The following three aims focus on addressing this issue.

1. Help develop distributed workflow software that capitalizes on existing cloud infrastructure to run efficiently at massive scale. Use this software to develop a robust, portable, open-source, and reproducible RNA-seq workflow to analyze 20,000 patient samples from four major studies. Make results available to the public through the UC Santa Cruz Genome Browser and UC Santa Cruz Xena platform.
2. Develop a scalable method for differential expression that produces results concordant to DE-Seq2 to efficiently process data from Aim 1. Comprehensively analyze gene and isoform expression across several cancers and their associated tissues to find repositional and novel drug targets.
3. Correct for batch effects in RNA-seq data using hierarchical Dirichlet processes. Extend the initial proposed model to eventually support gene-gene information.

1 Introduction

Efforts in genomic sequencing have advanced faster than Moore's Law, resulting in massive amounts of data as the cost to sequence an entire human genome plummets below the $1,000 mark (**Supplementary Figure 1**). Solutions to manage and analyze this ever-increasing data are imperative to continued progress in the field of genomics. Before discussing solutions in depth, an introduction which frames the problem and explains terminology is required.

Almost all known life is comprised of cells whose functions are governed by information located in DNA which most commonly resides in the cell's nucleus. Sections of DNA, called genes, are first transcribed into RNA, an almost identical copy that is capable of leaving the nucleus, which is then translated into what eventually become proteins. Proteins are the workers of the cell — responsible for structural, functional, and regulatory activity within the cell. Sequencing allows the information stored in DNA, made up of only four unique molecules called nucleotides, to be decoded. The human genome is about 3 billion nucleotides long and retrieving all of this information without complication is difficult.

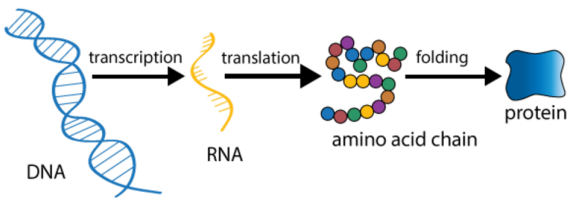


Figure 1: The central dogma is the process by which DNA is transcribed into RNA and then translated into proteins.

DNA sequencing can be understood as a series of simple steps. First, a biological sample is collected from a patient and the DNA is isolated and broken up into segments called reads. After some processing, the DNA is fed into a sequencing machine which outputs the sequence identity of the reads. The sequence identity is typically stored as letters representing the four nucleotides in DNA: A, G, T, and C. Effective techniques to isolate and sequence RNA have also been invented within the past decade. RNA is almost an exact copy of DNA, which begs the question: what additional information is gleaned from sequencing RNA that isn't provided by DNA sequencing?

RNA sequencing (RNA-seq) reveals the presence and quantity of RNA in a biological sample at a given moment [1]. The amount of DNA in a cell is typically static until cell division, but RNA is in a constant state of flux as the needs of the cell change for a variety of reasons. Since RNA is translated into protein, the relative abundance of RNA can be used to approximate the amount of each different protein created by a cell. This is valuable for studying how the cell environment is affected by drugs, environmental conditions, and diseases such as cancer. Cancer causes massive disruptions in the cell environment which lead to changes in gene expression. Understanding differences in gene expression between normal and cancer cells aids the development of treatments and helps scientists understand the mechanisms that allow cancerous cells to be so effective at replicating, avoiding apoptosis signalling, and evading the immune system. Given RNA-seq's utility for investigating disease, the next step is finding suitable RNA-seq datasets and determining how to process them at scale.

2 Rapid Large-scale Compute of 20,000 Patient Samples

Figure 2 depicts four different RNA-seq datasets totaling ~20,000 samples. Three of the four are cancer datasets, including The Cancer Genome Atlas (TCGA) which includes over 11,000 patients across 33 tumor types and represents the largest tumor collection of tumor data [2]. GTEx is the only non-cancer dataset and will be discussed in detail in Aim 2. All four datasets together represent more than 110 terabytes of patient data — more data than can fit on most machines and far too much data to process efficiently on most hardware available to academics. How can so much data be processed in a consistent and reproducible way?

Starting in late 2015, I began development on Toil: an open-source pure-Python workflow engine capable of massive scale [3]. Toil was written in Python given its popularity as an accessible open-source language that allows for easy distribution of packages through PyPI, a repository of software for the Python programming language. Toil is both robust and reliable due to its ability to resume a workflow from an arbitrary combination of leader or worker machine failures. Toil implements several optimizations to support effective scaling, and it supports all major commercial cloud providers. Unlike almost all other workflow engines [4–7], Toil supports both static and dynamic job scheduling, which complements the stochastic nature of many biological problems. Dynamic job scheduling encourages flexible design — allowing workflow authors to circumvent excessively complicated "wiring" of directed acyclic graphs elegantly — using principles of functional programming. A detailed explanation of Toil can be found in **Toil Architecture**. Toil provides the tools necessary to develop genomic workflows that can leverage commercial cloud environments for scale, but does nothing on its own to ensure that workflows output reproducible results that can be relied on regardless of what underlying infrastructure the workflow is run on. To encourage users to adopt reproducible practices into their own workflows, I wrote support into Toil for a program called Docker.

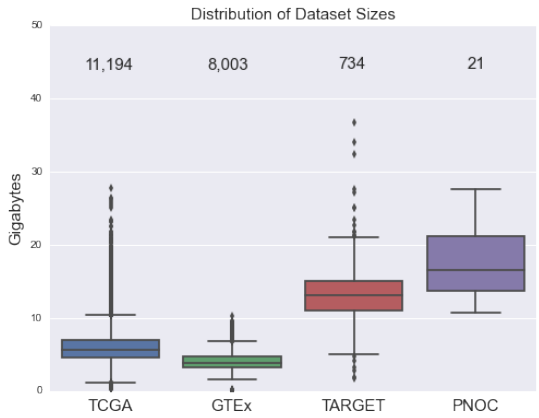


Figure 2: RNA-seq datasets chosen for large-scale compute. The Cancer Genome Atlas (TCGA), the Genotype Tissue Expression Consortium (GTEx), Therapeutically Applicable Research to Generate Effective Treatments (TARGET), and Pacific Pediatric Neuro-Oncology Consortium (PNOC) [2, 8–10]

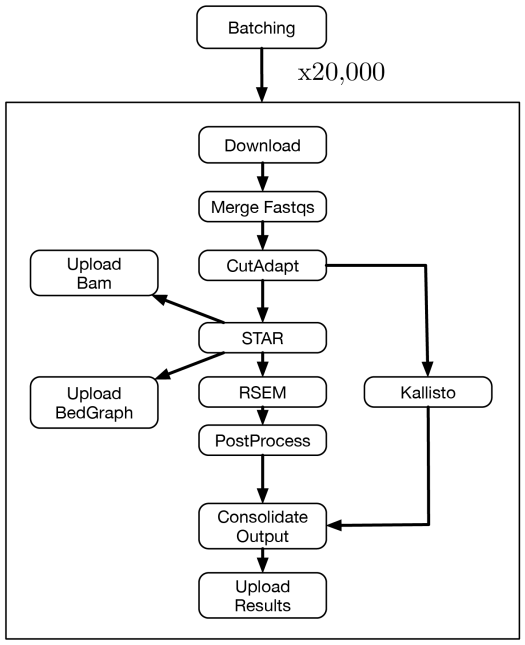


Figure 3: Dependency graph of the `toil-rnaseq` workflow as it was used in the large-scale compute of ~20,000 samples. See `https://github.com/BD2KGenomics/toil-rnaseq` for latest code.

Docker is a virtualization software that allows any software, such as genomic tools, to be encapsulated within an immutable environment such that it will deterministically provide the same output given the same input — disregarding stochastic elements within the software itself [11] (**Tool Containerization Philosophy**). Additionally, Docker images are easy to distribute and hashes of those images can be used

to ensure the proper image is always being used [12,13]. This gives researchers confidence the results they are using do not contain computational batch effects which result from differences in software versioning, dependencies, and environment [14]. Deterministic output is a critical achievement as researchers have traditionally used data produced from workflows that are prone to computational batch effects. For example, TCGA first processed all its RNA-seq data by distributing instructions to multiple institutions that included versions of tools to use and versions of some of those tools' dependencies. This leaves many avenues for batch effects to be introduced: user error in installing the proper tool version, version differences in dependencies that weren't listed, and operating system differences. Given these issues, I rewrote the workflow to take advantage of Toil's benefits like automation, parallelization, and the reproducibility offered by Docker.

TCGA's RNA-seq workflow has several components that can be broken down into four major steps: preprocessing, alignment, postprocessing, and expression estimation [15]. Preprocessing handles downloading and preparation of the sample data, alignment finds where in the genome those individual pieces of RNA came from, postprocessing prepares the data so it is in the proper format for the next step, and expression estimation takes the alignment information and estimates the expression value for every gene or transcript in the genome. I inherited this workflow as a collection of bash, Python, and Perl scripts spanning over 10,000 lines of code, with descriptions of what tools to install and their corresponding versions. Additionally, the workflow was not fully automated and required manual intervention on behalf of the user at least twice. Rewriting the workflow in Toil was a definite improvement, but unfortunately the workflow was still slow due to a bottleneck in the alignment step.

As a response to multiple issues with the TCGA RNA-seq workflow, I wrote my own workflow that no longer has a postprocessing step after alignment, produces expression estimates with more than one tool, is completely modular, uses Docker encapsulated tools, and is almost an order of magnitude faster than the TCGA RNA-seq workflow [16] (**Toil RNA-seq Workflow**). The primary speedup comes from substituting the slow alignment tool in the TCGA workflow for one that performs all of the alignment operations in memory [17]. An added benefit of this alignment tool is the output can be fed directly into the expression estimation tool which removes the need for the five postprocessing steps in the TCGA workflow. I also added tools that evaluate the quality of the input samples [18] and ensure the ends of the sequences do not erroneously contain adapters, which are small oligomers needed for sequencing but are sometimes accidentally included as part of the target sequence [19]. The efficiency and reproducibility of this workflow, coupled with the scalability provided by Toil, provides the means to process large amounts of RNA-seq data, but how can the data be processed for the lowest possible cost in a timely manner?

Researchers traditionally use servers that are owned by their university, company, or lab. These servers are often shared amongst a large pool of users, limiting the amount of resources available at any given time. Cloud computing allows renting of computational hardware like servers and data storage. Costs are only accrued while the hardware is in use, which is suitable for expedient large-scale computations that temporarily require lots of hardware but without the impracticality of having to permanently buy the equipment which is both expensive as well as wasteful. Not only can Toil perform distributed computation across multiple cloud machines, but because failures can be resumed, costs can be reduced even further through the use of preemptable servers like Amazon's Spot Market. Servers can be provisioned from the spot market for 50-80% less than the typical cost by letting users place a bid for the maximum amount they are willing to pay for the server. The caveat is that if Amazon's bid price exceeds the bid placed when the server was provisioned, the machine is terminated and everything that was on the machine is lost. This considerable downside is lessened by Toil's resume feature, and costs are negated entirely if the workflow consists of fine-grained tasks that each run in less than an hour due to Amazon's pricing scheme. After being granted a request to increase our provisionable limit to 1,000 machines, the Toil RNA-seq workflow was run with all 20,000 samples.

2.1 Results and Discussion

The workflow ran for about four days with a throughput of 99.6% and at its peak used 32,000 cores and 60TB of memory simultaneously. Over the four day period of the run, 3,267 unique nodes were provisioned, with a majority actively running for fewer than 5 hours before termination (**Figure 19**). To showcase that Toil can recover from failure, the entire cluster was brought down and subsequently resumed twice (**Figure 4**). Cost of the large-scale compute was calculated by querying spot price history for each node across its start and end time and summing to get $26,071, which was then divided by the 19,952 samples to get a per-sample cost of $1.30 — an estimated 30-fold reduction in cost and time compared to the TCGA RNA-seq workflow (**Large-scale Compute Logistics and Metrics**). The

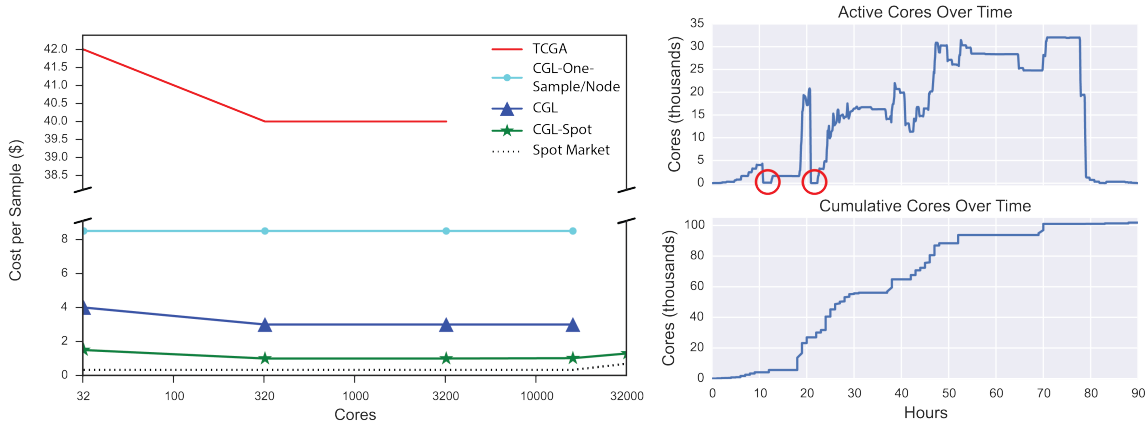


Figure 4: **Left**: Sample costs for different workflows and methods, with the cheapest method (green star) using the `toil-rnaseq` workflow on the spot market. **Right**: Number of cores and cumulative cores used over the course of the ∼4 day compute. The red circles indicate where the cluster was deliberately brought down then subsequently resumed.

output was stored in Amazon's Simple Storage Service (S3), before being converted into large matrices and made publically available through the UCSC Xena Browser [20]. A preprint of this project was submitted to BioRxiv [21], which reported the preprint placed in the top 5% of all research outputs scored by Altmetric and received a "High Attention Score" in the 97th percentile. The paper was then submitted to Nature Biotechnology where it was published in May 2017 [22]. Validation of the workflow's output was achieved by randomly selecting 10,000 samples and 10,000 genes to create sample/gene pairs, then comparing the expression values of those sample/gene pairs to TCGA's results. Results show the two workflows are significantly concordant (Pearson R correlation of 0.98) and heteroskedasticity in the plot shows greater variance for lower-expressed values as expected (**Figure 18**).

While this large-scale compute was successful, I noted several things that could be improved to further reduce costs and increase efficiency. Partitioning the compute across regions provides access to isolated spot markets, which reduces or removes the possibility of causing an increase in spot price through competition with yourself. The data would also need to be partitioned across S3 servers in those regions since ingress and egress is free if S3 is colocated with the servers and the inner-datacenter speeds are much faster. Now that Toil has auto-scaling, taking advantage of this strategy would only require a wrapper that accepts local data and the Toil workflow to run, uploads the data across regions, spins up a cluster in each region, and runs the workflow. Forthcoming projects like The Human Cell Atlas [23] only emphasize the necessity of scalable analyses in genomics. Algorithmic improvements like STAR and Kallisto [24] alongside advances like distributed computing and GPU acceleration will help keep pace when the scale of analysis reaches the order of a billion or trillion single-cell RNA-seq samples.

3 Tissue-level Analysis of Gene Expression in Cancer

3.1 Scalable Differential Expression

Differential expression, the identification of down- and upregulated genes between two or more phenotypes, is a routine downstream analysis performed on RNA-seq data. Several software packages exist for this analysis [25–28], with DESeq2 being a common choice [29] due to its effective normalization method which factors in sequencing depth and adjusts for overdispersion [27]. DESeq2 models counts as a negative binomial distribution and estimates the significance of gene's differential expression by calculating if the fold change (FC) is greater than a certain threshold given the gene's estimated mean counts and dispersion (variance). While this method is effective for a small number of samples and technical replicates, the runtime scales nonlinearly with respect to the number of samples (**Figure 5**).

Partitioning a single two group DESeq2 comparison into a set of smaller comparisons that are then combined can increase the runtime of DESeq2 by up to 97% (**Pairwise DESeq2 Method**, **Supplementary Figure 4**). **Figure 6** shows that a PearsonR of 0.9 can be obtained with a partition size as small as ∼log2(5) and a PearsonR of >0.95 once the partition size is greater than ∼log2(6). An optimal

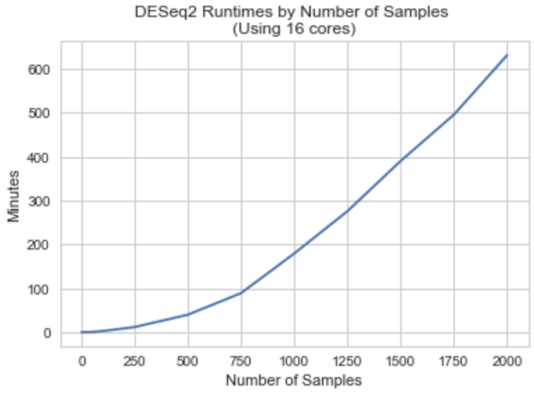


Figure 5: DESeq2 runtimes increase non-linearly as a function of sample size.

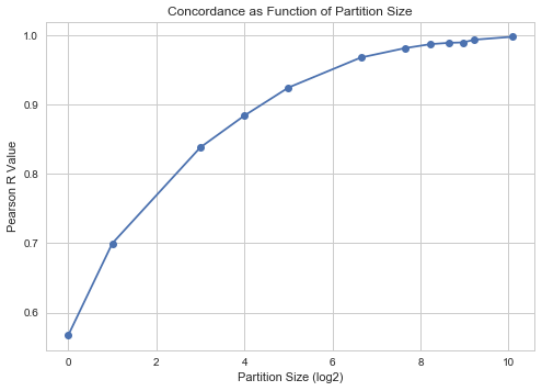


Figure 6: Gene rank concordance between traditional DESeq2 and the pairwise DESeq2 method rapidly increase as the number of samples in the partition reaches 32-256 samples.

partition size for speed and concordance is around ∼log2(7-9) depending on number of available cores and sample size.Given two groups, A and B, the optimal partition size $\hat{p}$ for a group given a maximum partition size p_M is calculated by taking the maximum of the sum of the partition size and its modulo component with respect to the size of the group N.

$$f(N,p) = \max_{i \in p_M} \sum_{i=1}^{p_M} \begin{cases} 2i, & \text{if } (N \bmod i) = 0 \\ i + (N \bmod i), & \text{otherwise} \end{cases}$$

The complexity is $O(n)$ but can be terminated earlier than n if the iteration of values in the vector $\{1, \cdots, p_M\}$ is inverted, because then the scoring function will never be larger than the prior maximum score after decreasing a single time. A comprehensive assessment across larger comparisons — to measure the relationship between variance and sample size needed to get an accurate recapitulation of the traditional comparison — is planned to measure of the method's efficacy. This method allows scaling of differential expression which means the large dataset computed in Aim 1 is now more manageable to analyze.

3.2 GTEx as a Substitute for TCGA Normals

Previous large scale comparisons of gene expression in TCGA have been done on the order of <5,000 samples with a majority (∼90%) of the samples derived from primary tumor tissue and the rest a handful of "normal" tissue, some of which is taken from the same patient carrying the tumor [30, 31]. In one study, the dataset consisted of 4,043 tumor samples and 548 normal tissue samples across 21 TCGA cancer types — an average of only ∼26 normal samples for each cancer type compared to ∼200 tumor samples [31]. While gene expression in normal tissue is more homogeneous than tumor samples, batch effects and contamination are a common problem with RNA-seq, which complicates an already noisy data source. Additionally, the small sample sizes cannot accurately reflect the general population, and therapeutics guided by results obtained from these analyses may have a higher likelihood of failing in clinical trials.

Given the lack of normals in TCGA, the second largest dataset processed in the large-scale compute are non-cancerous samples collected from The Genotype Tissue Expression Consortium (GTEx), which provides valuable insights into the mechanisms of gene regulation by studying human gene expression and regulation in tissues from healthy individuals [8]. Combining GTEx and TCGA and plotting the distribution of samples across tissues emphasizes how few normal samples are available to researchers only using TCGA data (**Figure 7**). Metadata information, combined with a novel clustering method developed at UC Santa Cruz [32], guided the grouping of TCGA and GTEx samples by tissue type. For brevity, TCGA tumor and normal samples will be referred to as Tumor and Normal.

Characteristic of most biological data, the TCGA and GTEx dataset is nosisy and samples from the same tissue between GTEx and TCGA don't necessarily cluster well (**Figure 8**). To get an idea of which tissues are suitable for comparison, the differences between gene fold change values when comparing

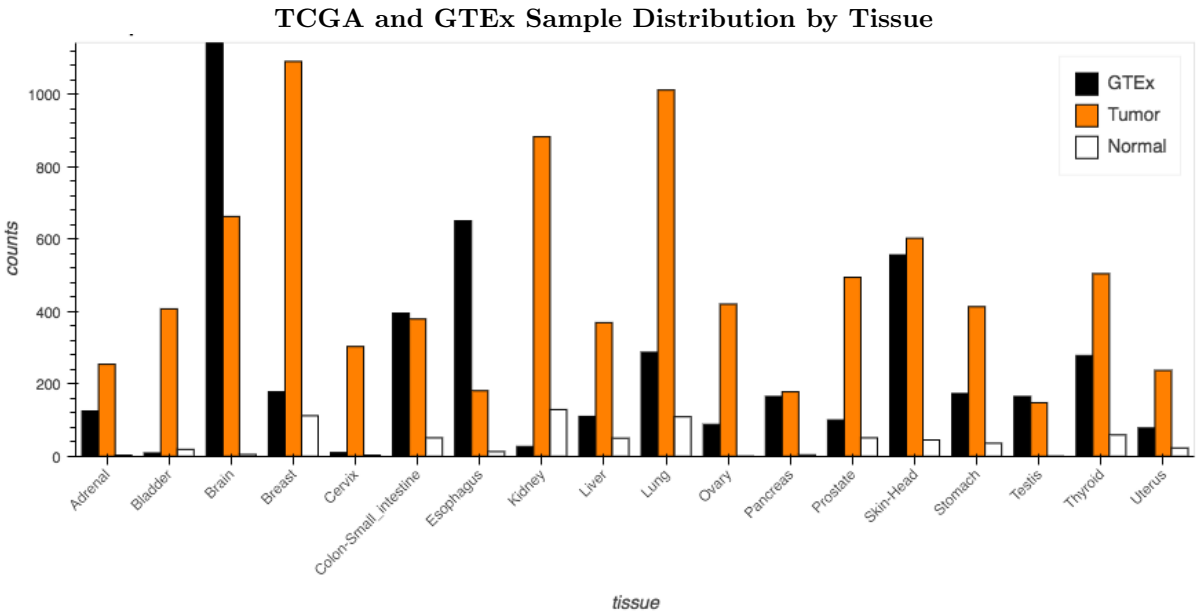


Figure 7: Bar graph emphasizing the low number of normal samples in TCGA relative to GTEx and TCGA tumor samples.

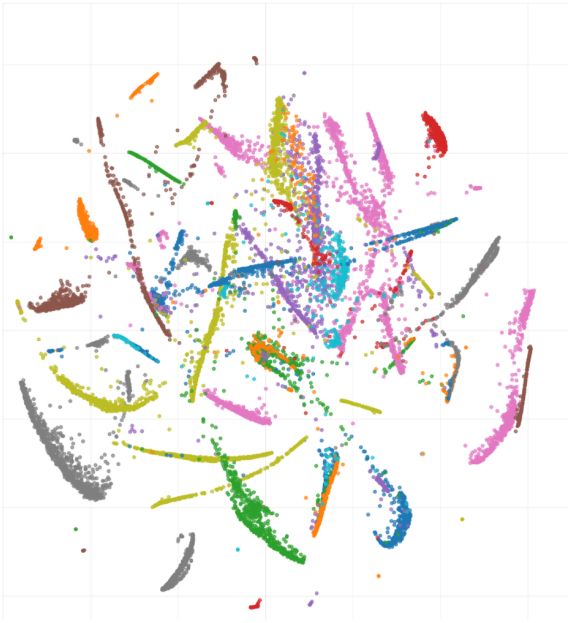


Figure 8: Gene expression visualization of TCGA and GTEx using t-SNE [33].

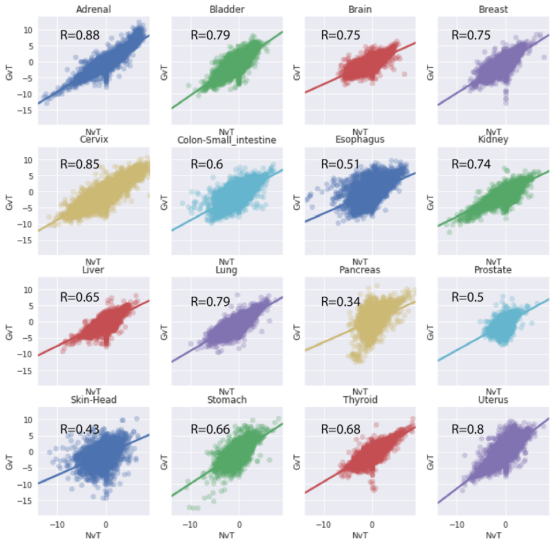


Figure 9: Gene fold change concordance between GTEx/Tumor and Normal/Tumor across 16 tissues.

GTEx and Tumor to the values obtained from comparing Normal to Tumor were plot against each other and pearson correlation values were calculated (**Figure 9**). To validate this data, a negative and positive control experiment to find known drug targets for treating cancer was set up. These drug targets look for proteins expressed on the surface of the cell, which to be effective targets must be highly expressed in the tumor cell but lowly expressed in normal healthy tissue.

The expression matrix was prepared by downloading RSEM [34] gene expression tables for both TCGA and GTEx from UCSC Xena [20], retaining only protein-coding genes, filtering out samples not represented in the intersectional metadata, and reversing Xena's inherent $\log_2(x+1)$ normalization. Counts were normalized using DESeq2 to account for sequencing depth and overdispersion and samples were grouped by tissue type. Kidney was selected as an example tissue, and a negative control was run

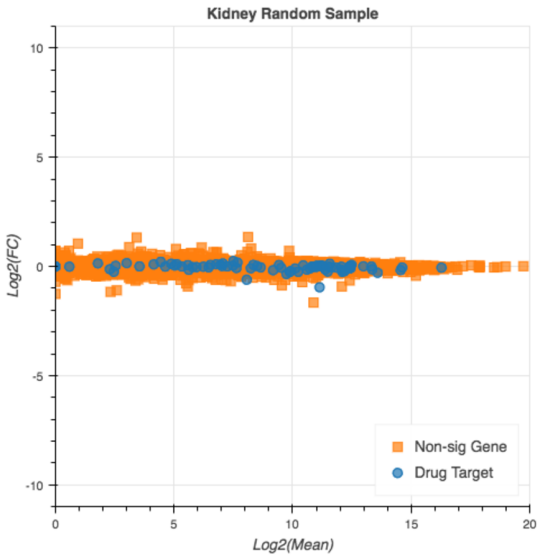


Figure 10: Negative control. Kidney samples were randomized before comparison. Y-axis is $\log_2$ fold change and x-axis is $\log_2$(concentration).

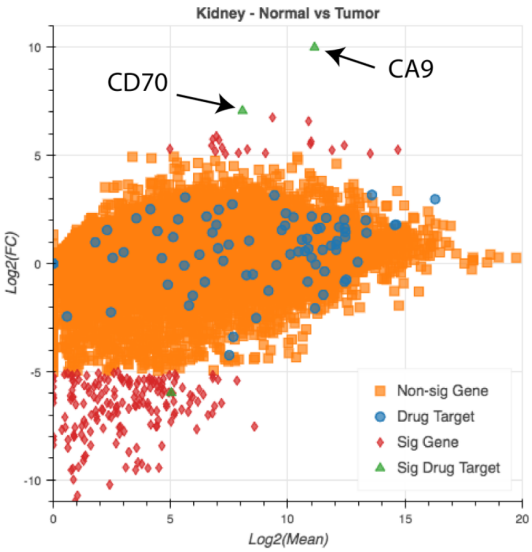


Figure 11: Positive control. 81 gene targets for cancer were labeled, with CD70 and CA9 being the only two specific to renal cell carcinoma.

by shuffling samples before visualizing as a Bland-Altman plot [35] which shows no significant up- or downregulation as expected (**Figure 10**). For the positive control, a set of 81 therapeutic monoclonal antibodies for cancer were identified and labeled, of which the two most upregulated are CD70 and CA9 – the only two in the set specific to renal cell carcinoma [36–39] (**Figure 11**, **Cell Surface Proteins and Drug Targets**).

Despite fold change values not being perfectly concordant to Tumor vs. Normal, the same overexpressed drug targets are found when comparing Tumor to GTEx. Plotting the distribution of expression across our datasets for these two genes shows the pattern of low expression in normal samples and high expression in tumor samples that indicate a cell-surface protein may be a candidate for drug targets. Successfully using GTEx as a substitute for Normal in this case provides evidence that GTEx tissues could potentially be used in cases where there are no Normal tissues in TCGA.

Future work on this project would: 1) attempt to understand how to confidently determine whether GTEx samples for a tissue are appropriate to use in comparisons to TCGA tumor samples of the same tissue type, 2) identify known cancer drug targets that have the potential to be repurposed in a different tissue (repositioning), and 3) discovery of possible novel drug targets by cross-referencing cell surface proteins obtained from the Cell Surface Protein Atlas [40] and looking for other drug targets like kinases and other targetable signaling molecules.

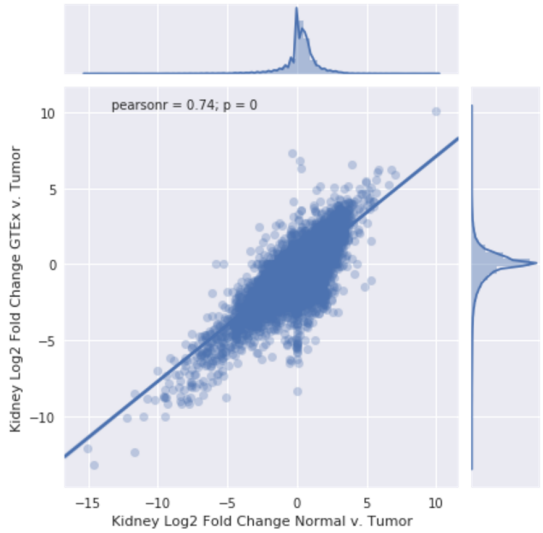


Figure 12: Spearman correlation plot of $\log_2$ fold change values when comparing Tumor to Normal and Tumor to GTEx.

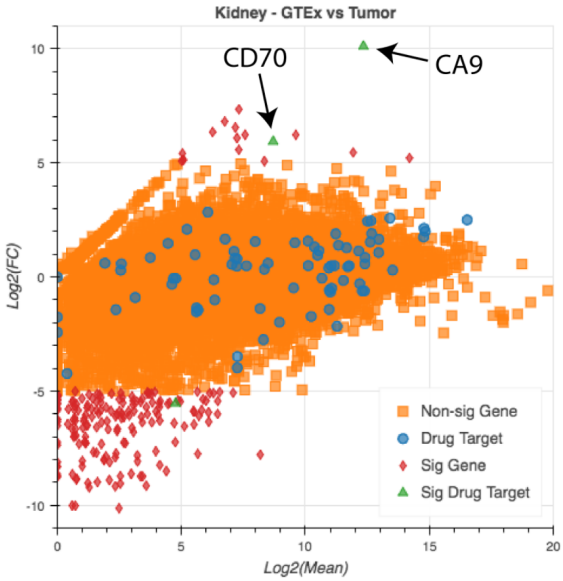


Figure 13: MA plot comparing GTEx to TCGA tumor. The same two gene targets (CD70 & CA9) that are upregulated when comparing TCGA normal and tumor are also identified.

4 RNA-seq Batch Effect Correction

Analyses like differential expression seek to identify the relevant biological differences between groups of samples against background signal noise such as inter- and intrasample variation, but some sources of variation, like batch effects, are unrelated to such variation [41]. Batch effects are technical sources of variation in sample data due to differences in laboratory preparation, experiment times, equipment, and technician error [42]. Batch effects have been reduced in recent years with the advent of high-throughput technologies that provide higher scalability, but generating the massive datasets needed to arrive at confident results for clinical practice requires not only multiple technicians and machines, but usually several different laboratories or facilities.

RNA-seq data is prone to batch effects due to the natural instability of RNA and the many steps required to go from human sample to sequenced reads. Methods for correcting batch effects in RNA-seq data are mostly repurposed algorithms used to correct batch effects in microarrays — 2D arrays that assay large amounts of biological material using multiplexed and parallel processing methods [43] — and include: distance-weighted discrimination [44] based on support vector machines, gene-wise one-way analysis of variance called mean-centering [45], surrogate variable analysis which combines singular value decomposition and a linear model analysis to estimate the eigenvalues from a residual expression matrix which already removed biological variation [46], geometric ratio-based method which scales sample measurements by the geometric mean of a group of reference measurements [42], and an empirical Bayes method, called Combating Batch Effects When Combining Batches of Gene Expression Microarray Data (ComBat) [47] that estimates parameters for location and scale adjustment of each batch for each gene independently [48]. Although all of these methods are algorithmic, there exist combination batch effect correction methods like factor analysis of spike-in controls [49], but when collating large datasets it is unlikely that all samples will share the same control methods. ComBat was found to be one of the most effective methods for microarrays [48], particularly when combined with quantile normalization of the separate batches before running ComBat [50]. Unfortunately, ComBat hasn't been supported in several years and fails to converge when run on a dataset as large as TCGA/GTEx, and since it wasn't originally designed for RNA-seq, it can produce negative values for expression counts.

With the rising popularity of neural networks and deep learning, it is surprising that the first paper to apply those approaches to batch effect correction was earlier this year [51]. This approach uses residual network blocks [52], which are standard ReLU / weight layers sandwiched between batch normalization layers that are trained via maximum mean discrepancy (MMD) — a measurement for distance between two probability distributions [53]. While the authors' preliminary results of single-cell RNA-seq show

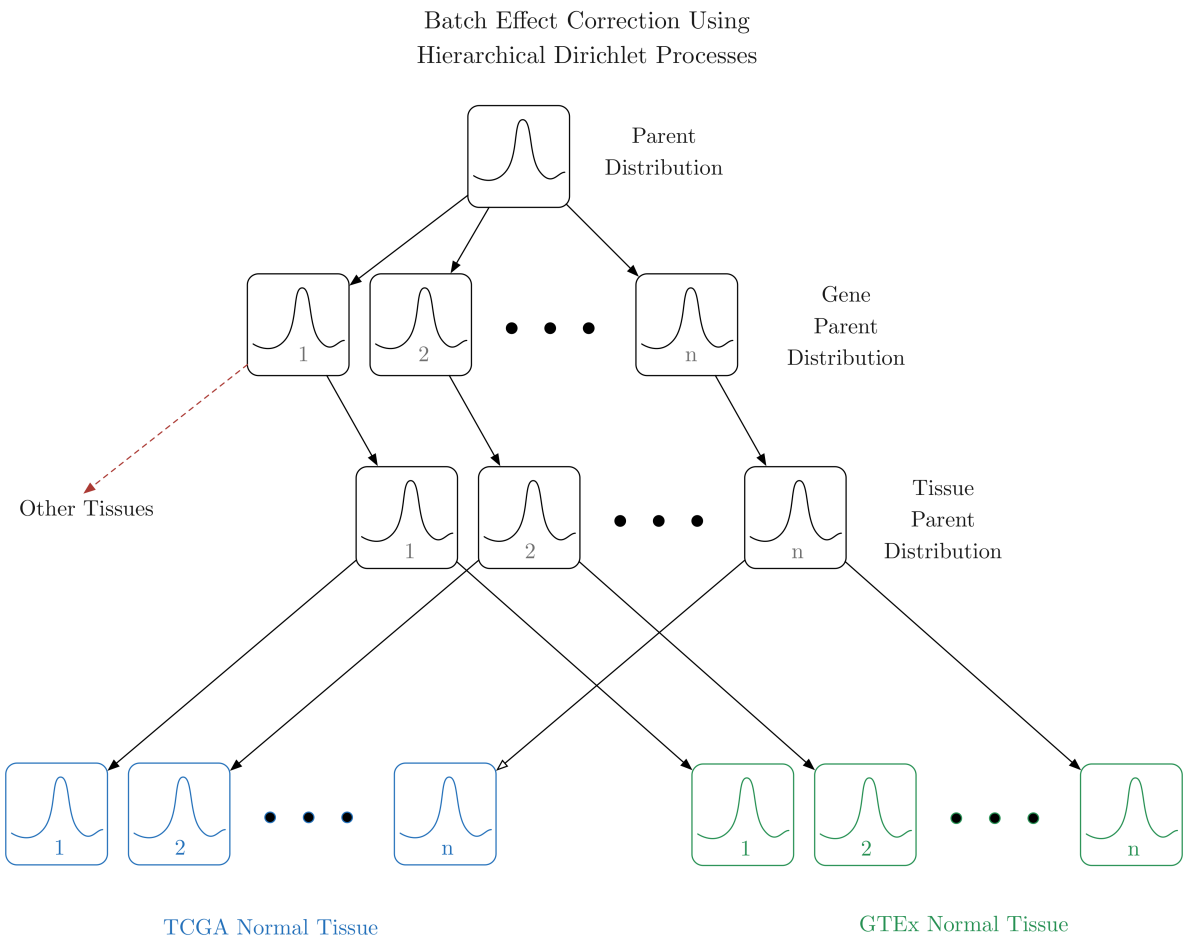


Figure 14: Modeling batch effects using a hierarchical Dirichlet process mixture model. Genes are modeled independently from a parent distribution which is then sampled for mixture components using a Dirichlet process to create downstream distributions.

what appears to be convincing batch normalization, a comparison to ComBat show only marginally better results. One problem with the results is all samples used were reference samples — that is, all samples were run in both batch A and batch B with the expectation that their distributions should be identical, and their method was not generalized to see how it performs on non-reference samples. In the discussion, the authors state that they plan on testing the efficacy of their method by "perform[ing] calibration in scenarios where multiple batches are present, each batch contains multiple samples and all batches contain a reference sample, [using only the reference samples for training but applying calibration to all samples] [51]." While the authors reasonably defended the choice of neural network and training methodology, their approach isn't tenable until follow-up experiments show it to be effective on real data.

One way to model batch effects would be to use a Hierarchical Dirichlet Process (HDP) mixture model — a collection of mixture distributions composed of atomic shared mixture components which themselves are distributed according to a Dirichlet process until eventually reaching the top-level parent distribution [54]. The proposed model would have a single parent distribution — likely a negative binomial distribution if counts are being modeled [27] — from which genes derive their own distribution using all samples in TCGA and GTEx, followed by a tissue-level layer derived from the gene parent distribution, with the leaves of the HDP representing TCGA normal and GTEx tissue (**Figure 14**). Sharing information in this way improves statistical strength, as the distributions end up more similar than if modeled independently [55]. Another benefit to this approach is an HDP library already exists that can be readily modified for this use case [56]. A downside is this model doesn't share information between the genes, which would reduce bias, and the HDP library isn't easily extendable to support gene-gene information. Refactoring the HDP library to add this support could potentially take significant development time.

There are many hurdles to overcome when considering how to correct for batch effects between TCGA and GTEx: no reference samples exist that were processed by both groups which can be used for calibration, GTEx samples were almost all collected from deceased individuals whereas TCGA normals were biopsied from mostly living tissue, they used different RNA-seq protocols for processing samples, there are few normal samples available in TCGA which renders most machine learning approaches unusable, and TCGA normals risk being unrepresentative of true normal tissue because most samples are obtained from a diseased organ which may be contaminated or phenotypically different due to stress caused by the disease. Despite these complications, attempts to correct for batch effects in RNA-seq data may significantly increase statistical power when using GTEx as a substitute for TCGA normals. If shown effective, this method may also be suitable for other groups that regularly use RNA-seq data for clinical projects, such as UCSC's Treehouse project.

5 Methods

5.1 Toil Architecture

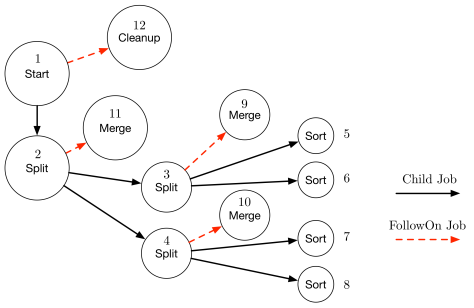


Figure 15: Merge sort using Toil's dynamic job scheduling capabilities.

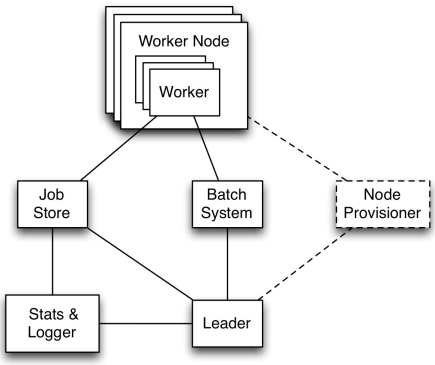


Figure 16: Diagram of Toil's Architecture.

A workflow comprises a set of tasks, or *jobs*, that are orchestrated by specification of a set of dependencies that map the inputs and outputs between jobs. This means workflows can be visualized as a graph, where the nodes represent *jobs* and the directed lines represent dependencies. Toil's atomic unit of work is the job, which is *checkpointed* after successful completion so that workflows can be restarted in case of failure and is most often written as a single Python function. Toil supports traditional static declaration of jobs (**Static Job Example 7.1**), but also boasts support for dynamic job generation (**Dynamic Job Creation 7.2**). For example, this means a workflow can be written such that a single job is run per-chromosome. Dynamic job declaration combined with Toil's use of *promises* and multiple job types encourages simplified and elegant workflow design (**Figure 15**). A *parent* job can have *child* jobs and *follow-on* jobs – child jobs are run following the successful completion of the parent job in parallel and follow-on jobs of a job are run after its child jobs and their successors have completed, also run in parallel. Follow-ons allow the easy specification of cleanup tasks that happen after a set of parallel child tasks. A parent job can pass the results of children jobs to follow-on jobs through the use of *promises* (**Promises 7.3**), which create a promised object that is later fulfilled upon completion of the child job. Toil also has a special *service* job for hosting long-running processes or services that other jobs must interact with during the course of a workflow, like a database or SPARK cluster (**Supplementary Figure 1**).

Toil's software architecture has six basic components: the leader, the batch system, workers, the job store, the statistics and logging monitor, and conditionally, the node provisioner (**Figure 16**). The leader is responsible for deciding which jobs should be run by traversing the job graph. Toil orchestrates running jobs on hardware using a modular *batch system* interface that supports a variety of systems such as Slurm, GridEngine, Mesos, Parasol, in addition to a built-in batch system for running on single machines like a laptop, which is useful for development and testing before deployment [57–60]. Workers are temporary processes responsible for running one job at a time. The worker monitors its job and reports back success or failure to the leader by editing the job's state in the job store. As jobs are run by the batch system, intermediate files and job information are stored atomically in Toil's job store, ensuring

the workflow can be resumed upon failure. Files are written to the job store in exchange for a FileStoreID object that can be passed to other jobs and later exchanged for a path to the file. Toil's default job store is a location on disk, but support also exists for Amazon's cloud storage, and comprehensive support for Microsoft Azure and Google Cloud are both in active development. The statistics and logging monitor collects information from the workers over the course of the run and distills it into a report, which describes both the overall run as well as individual jobs. The node provisioner is a recent feature that allows the creation of worker nodes which the batch system then includes in the pool of resources to which jobs are scheduled.

Toil implements several optimizations designed for scalability, including a read-only leader, job chaining, preemptable node support, and caching. Toil's leader process is currently constrained to a single thread, but steps were taken to ensure its not a bottleneck. Most of the leader's tasks revolve around processing the state of jobs, which are stored as a file within the job store. To minimize the load on this thread, each worker does as much as possible to manage the state of its own job. As a result, with a couple of negligent exceptions, the leader process never needs to write or update the state of a job within the job store. For example, when a job is complete and has no further successors the responsible worker deletes the job from the job store, marking it complete. The leader then only has to check for the existence of the file when it receives a signal from the batch system to know that the job is complete. This off-loading of state management is orthogonal to future parallelization of the leader. Scheduling of successor jobs is also partially managed by the worker, again reducing the number of jobs the leader needs to process. This job chaining heuristic is simple: if there is a single next successor job to run and its resources fit within the resources of the current job and closely match the resources of the current job then the job is run immediately on the worker without interaction with the leader. These optimizations cause the leader to use ∼4-6% CPU when running a workflow with ∼10,000 jobs and 10 worker nodes running ∼32 jobs at a time per worker. Critical to running at large-scale is dealing with intermittent node failures. Toil is therefore designed to always be resumable providing the job store does not become corrupt. This robustness allows Toil to run on preemptable nodes, which are only available when others are not willing to pay more to use them. Designing workflows that divide into many short individual jobs can capitalize on preemptable nodes allowing for workflows to be efficiently scheduled and executed. Running bioinformatic workflows often require the passing of large datasets between jobs. Toil caches the results from jobs such that child jobs running on the same node can directly use the same file objects, thereby eliminating the need for an intermediary transfer to the job store. Caching also reduces the burden on the local disks, because multiple jobs can share a single file. The resulting drop in I/O allows workflows to run faster, and, by the sharing of files, allows users to run more jobs in parallel by reducing overall disk requirements [3].

5.2 Tool Containerization Philosophy

The goal of encapsulating a genomics tool in a Docker container is to create a modular, portable tool that is software agnostic and can run on almost any hardware. The tool should be setup such that the call to the tool only requires the appended arguments prepended by the standard Docker boilerplate:

```
docker run quay.io/ucsc_cgl/<Tool> [Parameters]
```

1. The tool should be setup such that the Docker call to the tool only requires the appended arguments prepended by the standard Docker boilerplate. e.g.: `docker run quay.io/ucsc_cgl/<Tool> [Parameters]`
2. The Docker image should contain only the tool and the minimum dependencies needed to run that tool.
3. The tool should be launched when the person runs the image without needing to know where the tool is located or how it is called. If no parameters are passed, the user should be presented with the tool's help menu.
4. All images should have a folder **/data** that acts as the standard mount point. The final working directory in the container should be set to **/data** (`WORKDIR /data`).
5. Every image should have an `ENTRYPOINT` set to the tool's binary or wrapper script.

6. Tags will be used in two ways: to record information about that particular build of the image and for easy deployment. Our group uses Jenkins for continuous integration of the project and conforms to the following tag standard:
`{ToolVersion}--{MostRecentCommitHashForTool}`

5.3 Toil RNA-seq Workflow

The preceding issues with the TCGA mRNA-seq workflow spurned the development of a faster RNA-seq workflow with additional features. An obvious step missing from the TCGA mRNA-seq workflow is adapter trimming of the reads. When RNA is sequenced during RNA-seq, the resulting reads are usually longer than the RNA and therefore contain parts of the 3' adapter, which must be found and removed from each read before downstream in order to avoid errors [19]. Processing the same sample with and without CutAdapt results in 13,559 non-zero differences, with the largest difference being several thousand counts (**Supplementary Figure 2**) [61]. Correcting for low-end expression counts is becoming a more important issue with the rise of single-cell RNA-seq methodologies and the Human Cell Atlas – an initiative to profile a billion cells in the human body [23].

Another issue with the TCGA mRNA-seq workflow is a lack of Quality Control (QC) steps which provide information about the quality of the input FASTQs – this information is particularly valuable when examining unexpected or novel results. FastQC [18] produces a comprehensive report on base and sequence quality, sequence base and GC content, sequence length and duplication distributions, adapter content, and overrepresented sequences – often ribosomal RNA in RNA-seq. STAR [17], an alignment tool, also outputs log files with mapping information and other statistics that are saved as part of the workflow's standard output.

The workflow produces expression estimates using two different approaches: one approach derives estimated counts for transcripts directly from the sequencing reads themselves, and the other estimates counts after the transcripts have been aligned, specifically to the transcriptome of the reference genome. To understand how these tools work, a concrete definition of "read mapping" is necessary. A set of reads $\mathcal{F}$ with each read $F \in \mathcal{F}$ consisting of a sequence so that F is an ordered set $F = \{\theta_1, \cdots, \theta_l\}$, where $\theta \in \{A, C, G, T\}$. Denoting an ordered set of target sequences or contigs $\mathcal{T}$ with each sequence $T \in \mathcal{T}$, where each sequence is comprised only of values in θ. Read alignment in early read alignment tools consisted of a pair of maps. First $\psi : \mathcal{F} \to \mathcal{T} \bigcup \{\emptyset\}$ which specifies the contig each read maps to, or in case of an unaligned read, a mapping to $\{\emptyset\}$. The second mapping $\mathcal{L}_F : F \to \psi(F) \bigcup \emptyset$, which assigns each base in the read to a corresponding base in the contig (or to $\{\emptyset\}$ in case of a gap). Now all that's needed is a model and algorithm, which for several programs consisted of modified versions of the Smith-Waterman algorithm with an accompanying scoring scheme for matches and mismatches, and an affine gap penalty which penalizes starting a gap and extending a gap differently. This algorithm and model produce global alignments $\mathcal{L}_F$, where order is preserved between elements of F and $\psi(F)$ [62].

Kallisto [24] is a recent expression estimation program that achieves its notable speed by estimating counts directly from the sequencing reads. The key insight to Kallisto is that it only attempts to find *which* transcripts a read could belong to, not *where* inside the transcripts the read may have come from. This approach allows Kallisto to run >95% faster than other quantification methods that rely on alignment first, yet retains accuracy comparable to leading expression tools like RSEM [34]. This speed is achieved using a method the authors call 'pseudoalignment', which is read assignment to target sequences without any base-level sequence alignment. Pseudoalignment of a read to a set of transcripts T is a subset $S \subset T$ with out any specific coordinates mapping each base in the read to a specific position in each of the transcripts S, as is present in typical read alignment. Kallisto generates psuedoalignments for reads by hashing k-mers together with a colored de Bruijn graph of the transcriptome, where nodes represent k-mers and each transcript corresponds to a path [24]. Redundant nodes that map to the same transcript are skipped and the final set of transcripts the reads maps to is determined by taking the intersection of all transcripts mapped to. Finally, quantification of transcript abundances from the pseudoalignments is done using a likelihood function adapted for RNA-seq.

$$L(\alpha) \propto \prod_{f \in F} \sum_{t \in T} y_{f,t} \frac{\alpha_t}{l_t} = \prod_{e \in E} \Big(\sum_{t \in e} \frac{\alpha_t}{l_t} \Big)^{c_e}$$

Where F is the set of fragments, T the set of transcripts, l_t the effective length of transcript t, and $y_{t,f}$ is a compatibility matrix defined as 1 if f is compatible with t, otherwise it is set to 0. α_t is the probability of selecting fragments from transcripts, and c_e are the number of counts observed from equivalence class e [24].

Alternatively, RNA-seq quantification can be achieved by first aligning reads to the transcriptome and then expression abundance is derived from the aligned reads. The CGL RNA-seq workflow uses STAR [17] for alignment, and then passes the aligned reads to RSEM [34] for transcription. STAR was chosen for alignment because it is about a factor of 50 faster than Mapsplice [63] with similar accuracy and has the ability to output a file that can be fed directly into RSEM without any post-processing. The only notable downside to STAR in comparison to Mapsplice is that it uses uncompressed suffix arrays which it loads into memory – this translates into about 60GB of memory when aligning to the human genome. STAR works by finding the Maximal Mappable Prefix (MMP), which for a read sequence R, read location i and a reference genome G, $MMP(R, i, G)$ is the longest substring $(R_i, R_{i+1}, \cdots, R_{i+MML-1})$ that matches one or more substrings of G, where MML is the maximum mappable length [17]. STAR builds alignments from all reads that were aligned to the genome during the MMP phase by first clustering reads by proximity, creating a possible set of local alignments based on a standard scoring scheme for matches, mismatches, indels, and splice junction gaps, and then choosing the highest score as the best final alignment for a read [17]. These aligned reads are then fed into RSEM (RNA-seq by Expectation Maximization) which computes Maximum Likelihood (ML) abundance estimates using the Expectation Maximization (EM) algorithm. The probability a read is derived from a given transcript, θ_i for transcript i and θ_0 for unmapped reads, is computed during each iteration of the EM algorithm using a restricted set of transcripts derived from the alignment data [34].

5.3.1 Workflow Design Philosophy

The CGL RNA-seq workflow was designed for portability, reproducibility, modularity, and ease-of-use. Featuring detailed documentation, a wiki, and distribution through PyPi which allows installation from any computer with a valid Python installation and internet connection `pip install toil-rnaseq` [16]. Since all tools used in the workflow are encapsulated in Docker containers for portability and reproducibility, the only software dependencies required on the host machine or nodes is Python and Docker. The workflow is also completely modular, allowing users to select any combination of tools and output Like many workflows, the CGL RNA-seq workflow has a plethora of configurable options that dictate how the workflow runs. To minimize the number of command line arguments a user has to provide, the workflow generates an editable config and manifest for the user to set workflow options and provide sample information. The workflow's source code remains stable through the use of continuous integration which runs all tests in the repository and require that they pass before any code is merged, which includes concordance validation to ensure the output remains deterministic. To capitalize on services that rely on automating calls to Docker containers, the entire workflow is also released as a stand-alone Docker container.

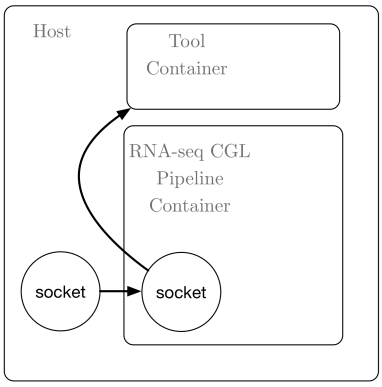


Figure 17: Diagram showing how the host Docker socket is mounted into the child container containing the `toil-rnaseq` workflow. When that child container executes commands to Docker containers, they are spawned by the host as siblings instead of nested children.

5.3.2 Docker-Encapsulated Workflow

Several genomics services [64, 65] work by running stand-alone Docker containers and by design are incapable of running the CGL RNA-seq workflow from its Python entrypoint. To accommodate these services, the workflow was encapsulated into a Docker image, but this caused performance issues as the parent Docker container was executing child Docker containers inside of the parent's virtual state. This issue was solved by mounting the host Docker socket into parent container's socket which means when the parent container executes a Docker container the host actually runs the container as a *sibling* to the parent container instead of nested within as a child process (**Figure 17**). Mounting the host socket into the parent container isn't problem-free, as it imposes a requirement that the host version of Docker's server matches the Docker client version installed in the parent container. To deal with this, the build process for the Dockerized workflow uses a templating scheme to iterate through a range of Docker versions and builds multiple containers.

5.3.3 Concordance

While the CGL RNA-seq workflow is about an order of magnitude faster than its TCGA counterpart, it is far less useful if it produces results that vary significantly. To validate concordance of expression values, 10,000 samples and 10,000 genes were randomly selected to create sample/-gene pairs. Expression values from TCGA for each sample/gene pair were collected by querying the UCSC Xena browser, and then plot against the expression value for the sample/gene pair computed with the CGL RNA-seq workflow. Results show the two workflows are significantly concordant (pearsonR of 0.98) and heteroskedasticity in the plot shows greater variance for lower-expressed values as expected (**Figure 18**).

To ensure stability of the CGL RNA-seq workflow, new versions of the workflow are validated only if all gene expression values are identical to earlier versions. This version concordance became standard practice after an innocuous change to STAR that keeps unmapped key pairs was found to cause downstream expression differences.

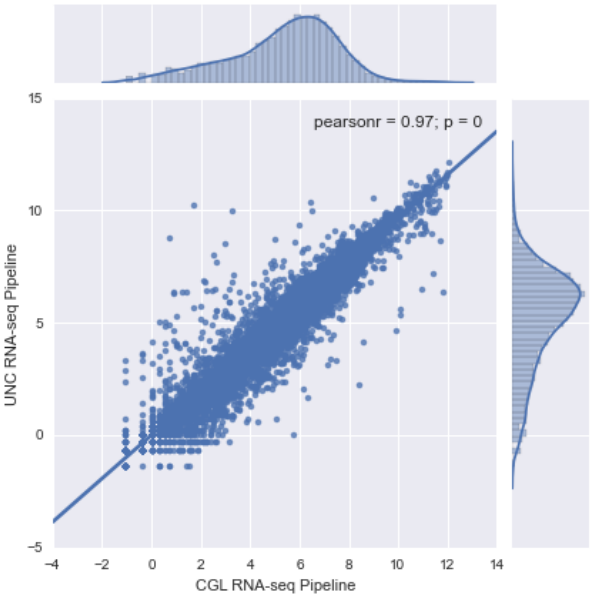


Figure 18: Concordance of gene expression between TCGA RNASeq Version 2 and CGL `toil-rnaseq` workflow.

5.3.4 Production Use

The CGL RNA-seq workflow has become the primary workflow run by the Genomics Institute Analysis Core [66] which provides sample analysis for Treehouse, a pediatric cancer research group at UCSC, as well as collaborators outside of UCSC. The CGL RNA-seq workflow has been run by several major institutions including: Children's Hospital of Philadelphia, California Kid's Cancer Comparison, West Coast Dream Team, Canada's Michael Smith Genome Sciences Centre, Genomic Data Commons Chicago, Treehouse Childhood Cancer Initiative, Broad Institute, and University of Washington [67–73]. Treehouse incorporated this gene expression data into their pan-cancer gene expression analysis alongside mutation data to nominate molecular pathways that may be driving the disease in each child, providing useful information to the medical teams [74].

The Python tool `vanity` shows that the CGL RNA-seq workflow has been pulled from PyPi over 6,000 times since the stand-alone repository was first established in September of 2016.

5.4 Large-scale Compute Logistics and Metrics

5.4.1 Datasets and Storage

To demonstrate Toil, the CGL RNA-seq workflow was used to compute gene- and isoform-level expression values for 19,952 samples from four studies: The Cancer Genome Atlas (TCGA), The Genotype Tissue Expression Consortium (GTEx), Therapeutically Applicable Research to Generate Effective Treatments (TARGET), and Pacific Pediatric Neuro-Oncology Consortium (PNOC) (**Figure 2**) [2, 8–10]. In preparation of the large-scale compute, a Toil workflow was written that downloaded, consistently formatted, and uploaded all samples to Amazon Web Services (AWS) Simple Storage System (S3) [75]. Storage in S3 was required because the transfer between S3 storage and AWS's Elastic Compute Cloud (EC2) is sufficient to not bottleneck the workflow as it scales horizontally across more nodes.

5.4.2 Metrics and Automation

EC2 metrics can be queried through Amazon's Cloudwatch except for memory and disk. At the time of this large-scale compute, EC2 clusters were being spun up using CGCloud [76] which spawns an "agent daemon" on every node that was originally responsible for managing SSH keys and privileges. I wrote a daemon thread that piggy-backs off deployment of the agent daemon to collect disk and memory usage as percentages via `psutil`, then injects these custom metrics into Cloudwatch at 5-minute intervals. To

minimize the cost of the compute, all EC2 nodes were provisioned from EC2's 'spot market', Amazon's name for preemptable nodes that are 50-80% cheaper than on-demand nodes, but if the market's current active bid price exceeds the user's bid price the node is terminated without warning. Toil is excellent for preemptable nodes as long as the workflow consists of mostly temporally fine-grained steps that run in less than an hour. I wrote automation code that spawned an initial cluster on the spot market, managed credentials, launched the CGL RNA-seq workflow, and then blocks while collecting metrics from EC2 in real time. Over the course of several hours (**Figure 4**), the number of nodes was increased by manually calling CGCloud's `grow-cluster` to periodically add 50-100 nodes until reaching 1,000 nodes, which provided a pool of 32,000 cores and 60TB of memory. This step was done manually as scaling to that size was untested and heuristics for managing cluster growth would have been made naively. For example, we did not foresee encountering the problem of having a direct impact on the spot market, and blindly growing the cluster – not realizing nodes were being removed due to competition with our earlier bids – would have contributed to a sizable increase in cost. As an additional cost-saving measure, idle nodes were terminated automatically during metric collection cycles as the workflow neared completion.

Metrics were parsed by creating a sparse matrix with nodes as rows and 5-minute interval timestamps as columns for each metric. At each timestamp, all nodes with values were used to compute mean and standard deviation (**Supplementary Figure 3**), and by counting all non-empty values for a timestamp column, the number of actively running instances – and consequently the number of cores – could be measured (**Figure 4**).

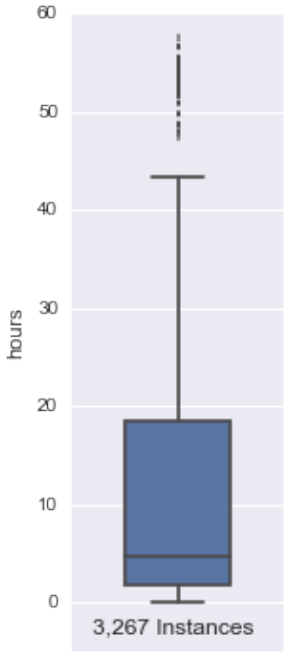


Figure 19: Distribution of time each instance used in the large-scale compute was active before termination. 3,267 unique instances were provisioned during the almost 4-day run.

5.5 Pairwise DESeq2 Method

Problem: DESeq2 is traditionally run as a single comparison between two groups. The input is an expression matrix of genes by samples, a vector identifying which samples belong to which group, and optionally, additional vectors describing covariate relationships in the data. Unfortunately, when trying to run a large number of samples, the runtime of DESeq2 increases exponentially with respect to the number of cores on the machine.

Breaking apart the sample sets and comparing these smaller partitions drastically reduces the runtime by making it both distributable as well as taking advantage of DESeq2's runtime at smaller sample sizes.

5.5.1 Partition Method

Breaking the comparisons into subgroups improves concordance, but in order to evaluate the method we need to know two things: the runtime at different subgroup sizes and the concordance to the traditional method.

Given two groups, A and B, where B is the larger group of size N, we can calculate an optimal partition size $\hat{p}$ from a vector of possible sizes $p = \{1, 2, \cdots, P\}$ by taking the maximum of the sum of the partition size and the modulo with respect to N.

$$f(N,p) = \max_{i \in p} \sum_{i} \begin{cases} 2i, & \text{if } (N \bmod i) = 0 \\ i + (N \bmod i), & \text{otherwise} \end{cases}$$

In case of ties, the max value calculated with the lowest partition size is used, as this minimizes the difference between the partition score and the remainder.

A distributed pipeline implementing this method can be found here.

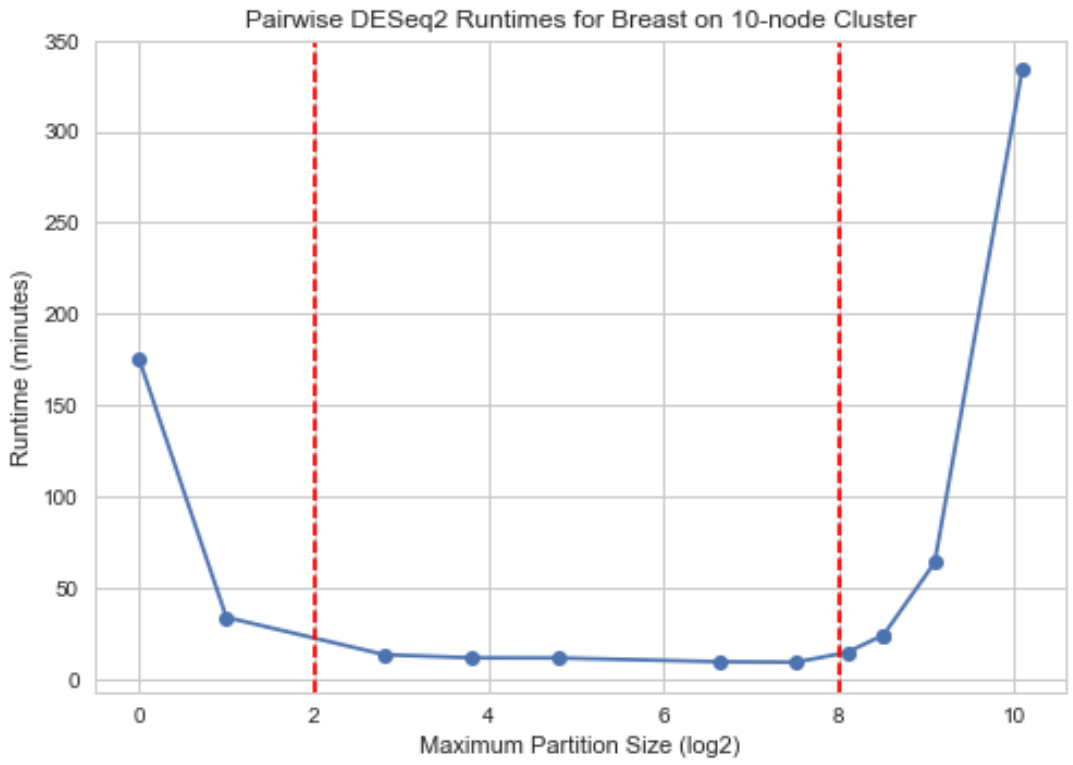
5.5.2 Runtime

- Tissue: Breast
- Normal: 113

- Tumor: 1092
- Hardware: 10-node cluster. 32 cores, 60GB of memory per node.

```
In [7]: partitions = [1, 2, 7, 14, 28, 100, 182, 273, 364, 364, 546, 1092]
        times = ['2:55:04', '34:08.96', '13:42.50', '11:59.06', '11:54.14',
                 '9:50.57', '9:35.78', '14:58.67', '24:30.76', '24:26.50',
                 '1:04:31', '5:33:21']
        new_times = []
        for time in times:
            time = time.split(':')
            if len(time) == 2:
                new_times.append(int(time[0]) + float(time[1]) / 60)
            if len(time) == 3:
                new_times.append(int(time[0]) * 60 + int(time[1]) +
                                 float(time[1]) / 60)
```

```
In [8]: plt.plot(np.log2(partitions), new_times, marker='o')
        plt.axvline(2, ls='--', c='r')
        plt.axvline(8, ls='--', c='r')
        plt.ylabel('Runtime (minutes)')
        plt.xlabel('Maximum Partition Size (log2)')
        plt.title('Pairwise DESeq2 Runtimes on 10-node Cluster');
```




```
In [ ]: partitions=[1, 2, 8, 16, 32, 128, 273, 546, 1092]
        runtimes = [57.5121666667, 23.1938333333, 11.0256666667,
                    9.853, 9.925, 5.40283333333, 13.5255, 62.0955,
                    327.7685]
        clocks = [47.3, 20.7, 6.6, 3.76, 2.5, 0.96, 0.55, 0.6, 0.88]
```

Traditional method takes by far the longest (5.5 hours), but the pairwise method with a partition of 1 also performs pretty poorly, which is fine because that method also produces results that are discordant from the traditional method. Our runtime optimum is reached with a partition size between 2^2 and 2^8 (4 - 256).

5.5.3 Concordance

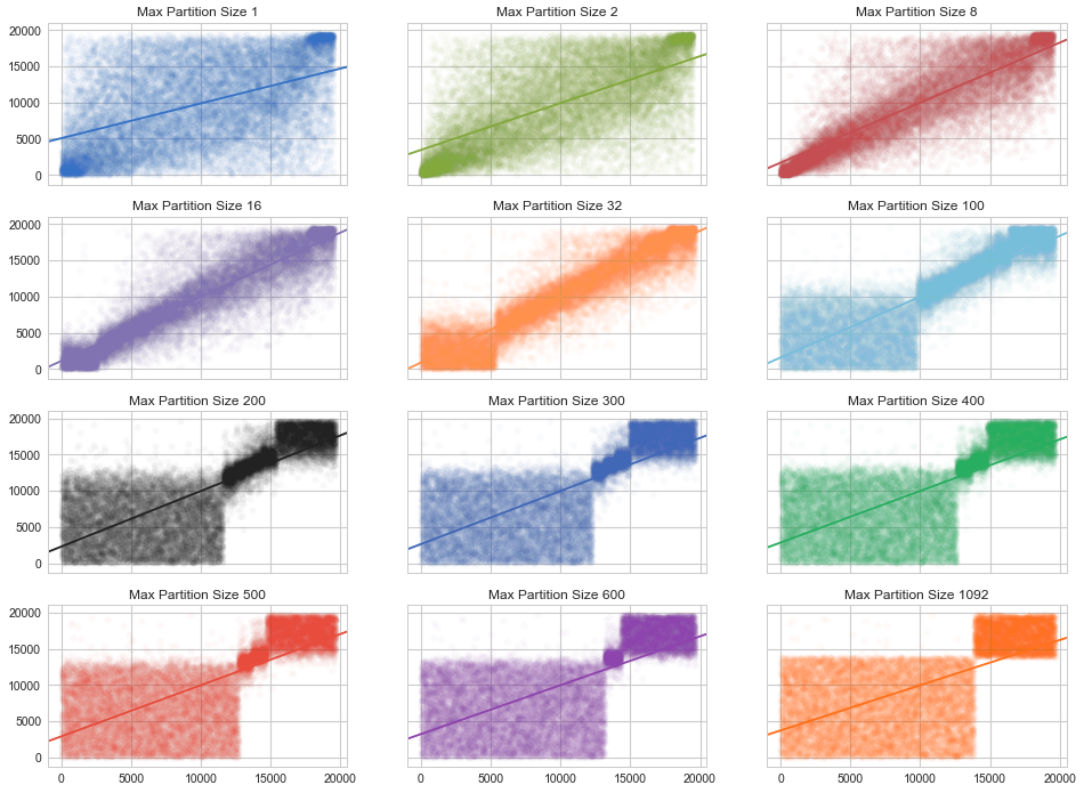
```
In [2]: def rank(ref_genes, genes_to_rank):
            temp = {}
            ranks = []
            inter = set(ref_genes).intersection(set(genes_to_rank))
            ref_genes = [x for x in ref_genes if x in inter]
            genes_to_rank = [x for x in genes_to_rank if x in inter]
            # print 'Length of intersection: {}'.format(len(inter))
            for i, gene in enumerate(ref_genes):
                temp[gene] = i
            for gene in genes_to_rank:
                ranks.append(temp[gene])
            return ranks

        def plot_ranks(ref_genes, genes_to_rank, title, ax):
            ranks = rank(ref_genes, genes_to_rank)
            x = np.array([x for x in xrange(len(ranks))])
            y = np.array(ranks)
            # sns.kdeplot(x, y, ax=ax)
            sns.regplot(x, y, ax=ax, scatter_kws={'alpha':0.03})
            ax.set_title(title)
```

```
In [3]: np_breast = pd.read_csv('nonpairwise-results/breast.tsv', sep='\t',
                                index_col=0)
        breast = {int(x.split('-')[1][:-4]): pd.read_csv(
                'max-chunk-results/' + x, sep='\t', index_col=0)
                  for x in os.listdir(
                "max-chunk-results/") if 'breast' in x}
```

Sorting by P-value Count

Given pairwise comparison, any gene considered significant is "binned", and the final set of genes are sorted by this "pvalue count", which intuitively represents the idea, "The number of times this gene was found significant across all comparisons."

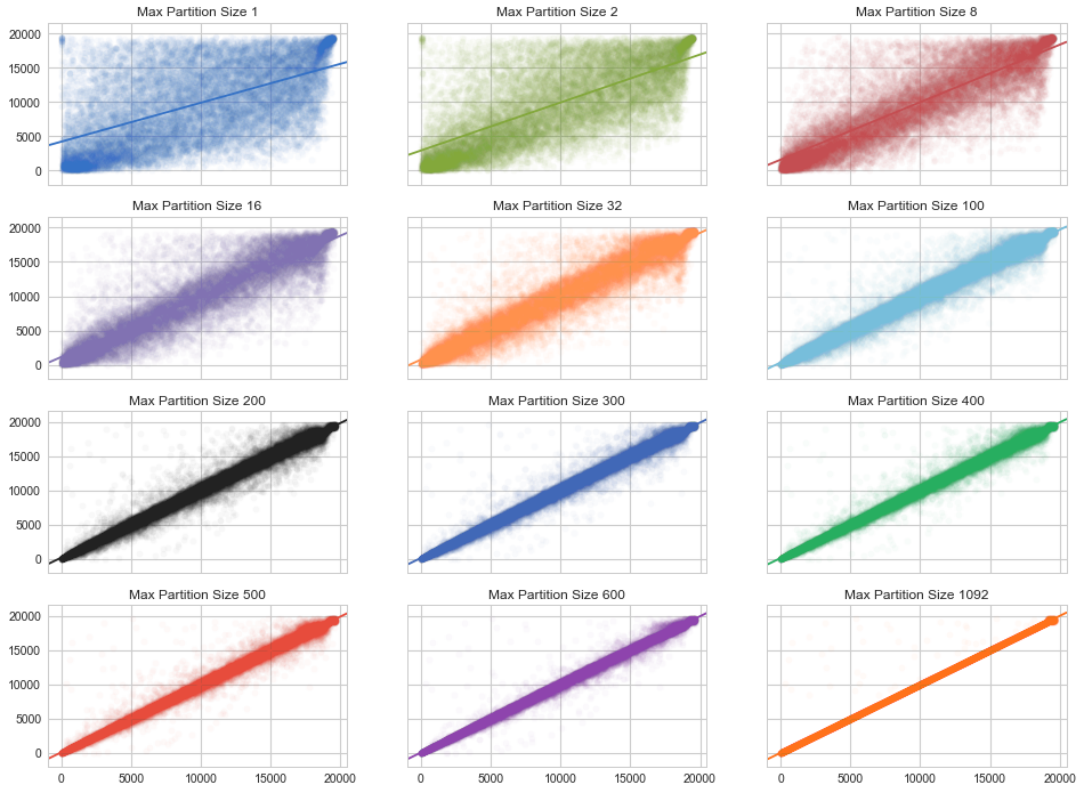
```
In [55]: f, axes = plt.subplots(4, 3, figsize=(16, 12), sharex=True,
                                sharey=True)
         axes = axes.flatten()
         for i, max_chunk in enumerate(
             sorted(breast.keys(), key=lambda x: int(x))):
             plot_ranks(np_breast.geneId, breast[max_chunk].index,
                        'Max Partition Size ' + str(max_chunk), axes[i])
```



Concordance when sorting by "Pvalue count" ends up with some bizarre results. Let's examine the same dataset, but sorting by averaged p-value.

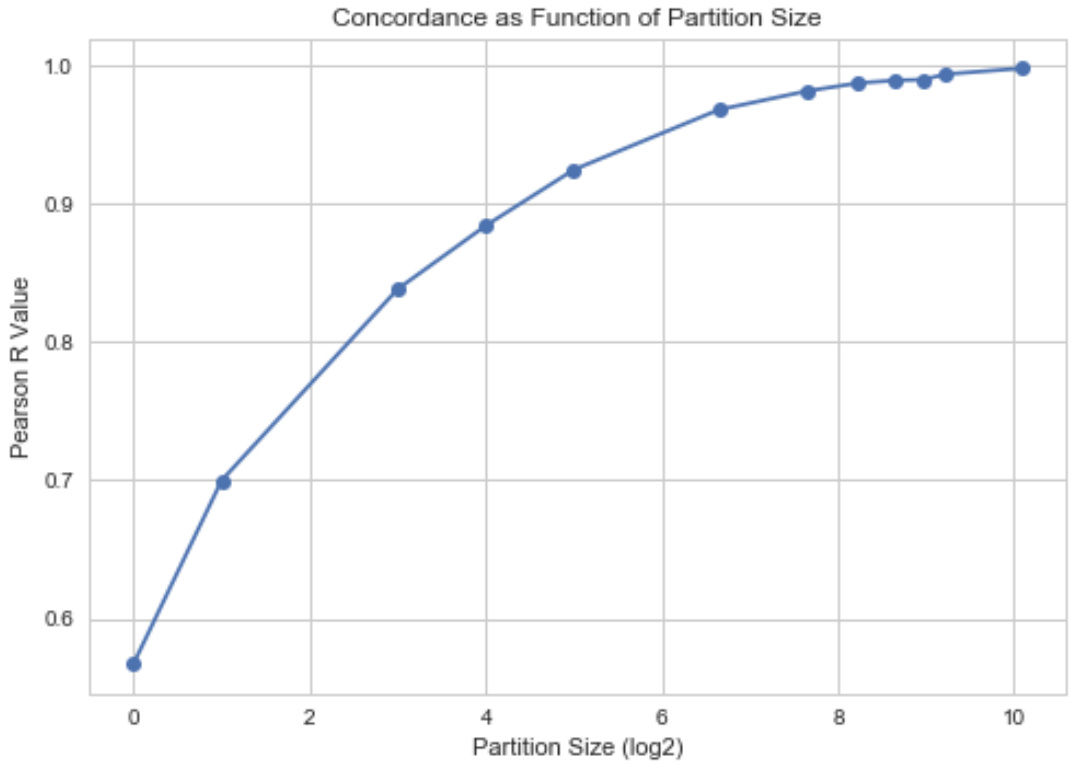
Sorting by Averaged P-value

```
In [56]: f, axes = plt.subplots(4, 3, figsize=(16, 12), sharex=True,
                                sharey=True)
         axes = axes.flatten()
         for i, max_chunk in enumerate(
             sorted(breast.keys(), key=lambda x: int(x))):
             plot_ranks(np_breast.geneId, breast[max_chunk].sort_values(
                     'pval').index, 'Max Partition Size ' +
                        str(max_chunk), axes[i])
```



Much better! Let's plot the pearson correlation as a function of the partition score

```
In [5]: ps = []
        for max_chunk in sorted(breast.keys(), key=lambda x: int(x)):
            r = rank(np_breast.geneId, breast[max_chunk].sort_values(
                    'pval').index)
            ps.append(pearsonr(np.array([x for x in xrange(len(r))]),
                               np.array(r))[0])
        plt.plot(np.log2([int(x) for x in sorted(breast.keys(),
                                                 key=lambda x: int(x))]),
                 ps, marker='o')
        plt.xlabel('Partition Size (log2)')
        plt.ylabel('Pearson R Value')
        plt.title('Concordance as Function of Partition Size');
```



At a partition size of ˜16, our results achieve a 0.9 Pearson R score compared to the nonpairwise method.

5.5.4 Pvalue Recombination with Weights

Example: - Group A: 100, Group B: 530 - Max-Partition size: 16 - Optimum: 14 - Remainder: 12 - (N_c) Number of groups size 14: 37 - (N_r) Number of groups size 12: 1 - N Number of total groups (38 in this example) - w = Vector of weights. 37 weights of (14/530) and one weight of (12/530) - p = Vector of pvals for each of the groups

Calculating the combined pval by weight is

$$\hat{p} = \sum_{i=1}^{N} w_i * p_i$$

or as vectors:

$$\hat{p} = \vec{w} \cdot \vec{p}$$

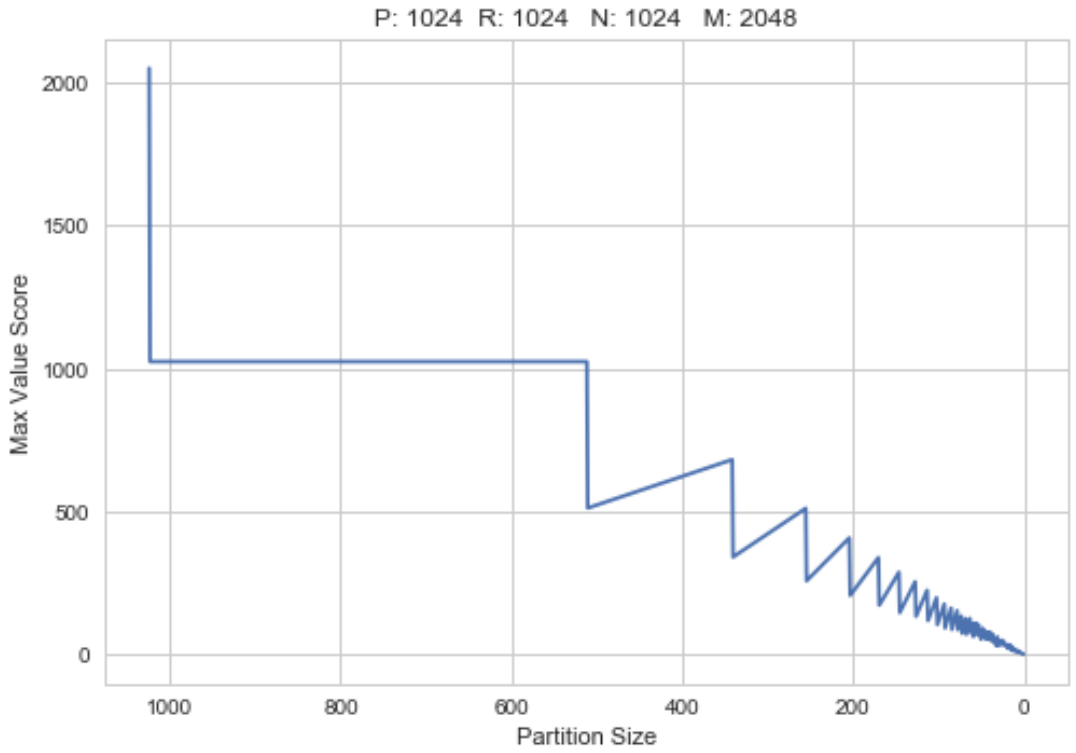
Max Value as a Function of Partition Score

To visualize how the partition size is determined given a group size N and maximum partition size P.

```
In [5]: def plot_max_val(N, max_partition):
            max_vals = []
            max_val = 0
            partition = None
            remainder = None
            temp = max_partition
            for i in xrange(temp):
                r = N % temp
                r = temp if r == 0 else r
                max_vals.append(r + temp)
```

```
            if r + temp >= max_val:
                max_val = r + temp
                partition = temp
                remainder = r
            temp -= 1
        plt.plot([x+1 for x in xrange(max_partition)], max_vals[::-1])
        plt.gca().invert_xaxis()
        plt.title('P: {}  R: {}   N: {}   M: {}'.format(
                partition, remainder, N, max_val))
        plt.xlabel('Partition Size')
        plt.ylabel('Max Value Score')
```

```
In [30]: plot_max_val(1024, max_partition=1024)
```



N of 530 and a max partition size of 16 gives us a maximum value of 26 when there are thirty-seven groups of size 14 and a single group of size 12.

5.6 Cell Surface Proteins and Drug Targets

Proteins expressed on the surface of cells can be used as targets for drugs like therapeutic monoclonal antibodies. The goal is to identify cancer subtypes where the tumor is upregulating a specific cell surface protein that is absent in healthy tissue of the same type.

```
In [4]: df_path = '/mnt/rna-seq-analysis/data/xena/deseq2_normalized_tcga_gtex_counts.tsv'
df = pd.read_csv(df_path, sep='\t', index_col=0)
```

Subset for CSPA genes

Map gene IDS to gene names

```
In [5]: gene_map_path = '/mnt/rna-seq-analysis/data/objects/gene_map.pickle'
gene_map = pickle.load(open(gene_map_path, 'rb'))
```

```
In [6]: genes = [gene_map[x.split('.')[0]] if x.split('.')[0] in gene_map
else x for x in df.index]
df.index = genes
```

```
In [7]: cspa_path = '/mnt/rna-seq-analysis/metadata/CSPA_genes.tsv'
cspa_genes = [x.strip() for x in open(cspa_path, 'r').readlines()]
print len(cspa_genes)
cspa_genes = [x for x in cspa_genes if x in df.index]
print len(cspa_genes)
```

```
1442
1343
```

5.6.1 Tissues and Metadata

```
In [8]: def get_tissues():
work_dir = '/home/ubuntu/rnaseq-recompute-analysis'
pair_file = os.path.join(work_dir, '/preprocessing/candidate_tissues.csv')
tissues = []
with open(pair_file, 'r') as f:
for line in f:
line = line.strip().split(',') if ',' in line else [line.strip()]
tissues.append(line)
return tissues

def flatten(x):
"""
    Flattens a nested array into a single list

    :param list x: The nested list/tuple to be flattened.
    """
result = []
for el in x:
if hasattr(el, "__iter__") and not isinstance(el, basestring):
result.extend(flatten(el))
else:
result.append(el)
return result

def get_l2fc(a, b):
return np.log2(b + 1) - np.log2(a + 1)

def get_samples_for_tissue(metadata, tissue):
"""
    metadata - DataFrame of intersectional metadata
    tissue - list containing all names associated with tissue
    """
name = '-'.join(tissue)
print 'Finding samples for tissue ' + name
return flatten([list(set(metadata[metadata.tissue == x].index)
         for x in tissue)])
```

```
In [9]: tissues = get_tissues()
```

```
In [10]: path = '/mnt/rna-seq-analysis/metadata/tcga_gtex_metadata_intersect.tsv'
metadata = pd.read_csv(path, index_col=0, sep='\t')
```

Plot CSPA Genes in All Tissues

```
In [51]: def make_plots(sub, subdir, d1, d2, n1, n2, name):
mean = sub.mean(axis=0)
```

```
d1_medians = sub.loc[d1].median(axis=0)
d2_medians = sub.loc[d2].median(axis=0)
l2fc = get_l2fc(d1_medians, d2_medians)
plot = pd.DataFrame()
plot['mean'] = np.log2(mean + 1)
plot['l2fc'] = l2fc
plot['gene'] = sub.columns
plot.index = sub.columns
bot25_genes = list(plot.sort_values('l2fc').index)[:25]

colorby = []
for i, gene in enumerate(sub.columns):
if gene in cspa_genes:
colorby.append('CSPA')
elif np.abs(l2fc[i]) > 5:
colorby.append('sig')
else:
colorby.append('unsig')

plot['colorby'] = colorby

tooltips = [
('Gene', '@gene'),
('Mean', '@mean'),
('L2FC', '@l2fc')]

p = Scatter(plot, x='mean', y='l2fc', title=name,
xlabel='Log2(Mean)', ylabel='Log2(FC)',
color='colorby',
tooltips=tooltips,
legend=True,
palette=Colorblind3,
responsive=True)
p.title.align = 'center'
output_file = os.path.join(subdir, 'MA', name + '.html')
save(p, output_file)

# Boxplots of top overexpressed (B respect to A)
top_genes = list(plot.sort_values('l2fc', ascending=False).index)
top_genes = [x for x in top_genes if x in cspa_genes][:25]
top1 = sub.loc[d1][top_genes]
top2 = sub.loc[d2][top_genes]

plot = pd.DataFrame()
plot['Log2(Counts)'] = map(lambda x: np.log2(float(x) + 1),
          flatten([top1.loc[x] for x in top1.index] +
                  [top2.loc[x] for x in top2.index]))
plot['gene'] = list(top1.columns) * len(top1.index) +
    list(top2.columns) * len(top2.index)
plot['dataset'] = [n1 for _ in xrange(top1.shape[0] * top1.shape[1])] + \
       [n2 for _ in xrange(top2.shape[0] * top2.shape[1])]

tooltips = [('# Samples', '@height')]
b = BoxPlot(plot, label='gene', values='Log2(Counts)', group='dataset',
tooltips=tooltips,
responsive=True,
title=name,
color='dataset',)
```

```
b.title.align = 'center'
output_file = os.path.join(subdir, 'Box', name + '.html')
save(b, output_file)
```

```
In [ ]: dirs = ['G_T', 'N_T', 'G_N']
for d in dirs:
if not os.path.isdir(d):
os.mkdir(d)
os.mkdir(os.path.join(d, 'MA'))
os.mkdir(os.path.join(d, 'Box'))

for tissue in tissues:
name = '-'.join(tissue)
samples = flatten([list(set(metadata[metadata.tissue == x].index)
            for x in tissue)])
samples = [x for x in samples if x in df.columns]
sub = df[samples].T
print 'Tissue: {}\tSamples: {}'.format(name, sub.shape[0])
g = list(set([x for x in samples if not x.startswith('TCGA')]))
t = list(set([x for x in samples if x.startswith('TCGA')
  and x.endswith('01')]))
n = list(set([x for x in samples if x.startswith('TCGA')
  and x.endswith('11')]))
if g and t:
make_plots(sub, 'G_T', g, t, 'GTEx', 'Tumor', name)
if n and t:
make_plots(sub, 'N_T', n, t, 'Normal', 'Tumor', name)
#if g and n and t:
#    make_plots(sub, 'G+N_T', g+n, t, 'GTEx+Normal', 'Tumor', name)
#    make_plots(sub, 'G_T+N', g, t+n, 'GTEx', 'Tumor+Normal', name)
if g and n:
make_plots(sub, 'G_N', g, n, 'GTEx', 'Normal', name)
```

Find Existing Drug Targets

1. Get list of drug targets from here
2. For each tissue or for the entire dataset:
    1. Find significantly upregulated genes that match the drug target
    2. MA Plot — color CSPA genes, drug target genes, and >log2FC genes

5.6.2 Identify Known Drug Targets

```
In [22]: url = 'https://en.wikipedia.org/wiki/List_of_therapeutic_monoclonal_antibodies'
dt = pd.read_html(url, header=0)
dt = dt[0]
```

Select drug targets used for cancer

```
In [23]: dt = dt[dt.Use.str.contains('cancer')]
```

```
In [ ]: targets = [x.split()[0].split('/')[0].strip() for x in dt.Target]
targets = set([x.replace('-', '').replace(',', '').upper() for x in targets])
```

Locate drug targets not present in our gene names and attempt to fix

```
In [13]: with open('raw_targets', 'w') as f:
for t in targets:
if t not in gene_map.values():
try:
```

```
 f.write('{}\tFIX\n'.format(t))
except UnicodeEncodeError as e:
 f.write('TGF\t{}FIX\n')
else:
f.write(t + '\n')
```

After laboriously fixing all the mismatched genes by cross-referencing GeneCards, read the results back in

```
In [11]: targets = [x.strip() for x in open('cleaned_targets.txt', 'r').readlines()]
targets = set(targets)
print '{} drug targets'.format(len(targets))
```

```
81 drug targets
```

Generate Negative / Positive Control MA Plots for Identifying Drug Targets

Given a tissue, generate a negative control by randomizing the two groups to be compared (use same y_range between both plots)

```
In [74]: def MA_plots(df, tissues, metadata, comparison='NvT', matched=False):
for tissue in [['Adrenal'], ['Brain'], ['Ovary'], ['Pancreas'], ['Testis']]:
name = '-'.join(tissue)
print name
samples = flatten([list(set(metadata[metadata.tissue == x].index)
                 for x in tissue)])
samples = [x for x in samples if x in df.columns]

t = list(set([x for x in samples if x.startswith('TCGA')
       and x.endswith('01')]))
if comparison == 'GvT':
normal_label = 'GTEx'
g = list(set([x for x in samples if not x.startswith('TCGA')]))
else:
normal_label = 'Normal'
g = list(set([x for x in samples if x.startswith('TCGA')
           and x.endswith('11')]))
if matched:
 comparison += '-Matched'
 # Select only samples for which there is a corresponding normal
 combined = set(x[:-3] for x in t).intersection(set(x[:-3]
                                                    for x in g))
 t = [x + '-01' for x in combined]
 g = [x + '-11' for x in combined]

if not g or not t:
print 'Not enough samples for: ' + name
continue

print '\tNumber of samples\tt: {}\tn: {}'.format(len(t), len(g))

# Create random groups 1 and 2
f1 = sample(g + t, len(g+t) / 2)
f2 = list(set(g + t) - set(f1))

print '\tCreating Dataframe'
sub = df[g + t].T

neg_plot = pd.DataFrame()
reg_plot = pd.DataFrame()
```

```
mean = sub.median(axis=0)
neg_plot['mean'] = np.log2(mean + 1)
reg_plot['mean'] = np.log2(mean + 1)

g_medians = sub.loc[g].median(axis=0)
t_medians = sub.loc[t].median(axis=0)
reg_l2fc = get_l2fc(g_medians, t_medians)
reg_plot['l2fc'] = reg_l2fc
reg_plot['gene'] = sub.columns

f1_medians = sub.loc[f1].median(axis=0)
f2_medians = sub.loc[f2].median(axis=0)
neg_l2fc = get_l2fc(f1_medians, f2_medians)
neg_plot['l2fc'] = neg_l2fc
neg_plot['gene'] = sub.columns

print '\tLabeling Samples'
reg_colorby, neg_colorby = [], []
for i, gene in enumerate(sub.columns):
if gene in targets:
 if np.abs(reg_l2fc[i]) < 5:
     reg_colorby.append('Drug Target')
 else:
     reg_colorby.append('Sig Drug Target')
else:
 if np.abs(reg_l2fc[i]) < 5:
     reg_colorby.append('Non-sig Gene')
 else:
     reg_colorby.append('Sig Gene')

for i, gene in enumerate(sub.columns):
if gene in targets:
 if np.abs(neg_l2fc[i]) < 5:
     neg_colorby.append('Drug Target')
 else:
     neg_colorby.append('Sig Drug Target')
else:
 if np.abs(neg_l2fc[i]) < 5:
     neg_colorby.append('Non-sig Gene')
 else:
     neg_colorby.append('Sig Gene')

reg_plot['colorby'] = reg_colorby
neg_plot['colorby'] = neg_colorby

tooltips = [
('Gene', '@gene'),
('Mean', '@mean'),
('L2FC', '@l2fc')]

print '\tBuilding Plots'
p_reg = Scatter(reg_plot, x='mean', y='l2fc',
         title='{} - {} vs Tumor'.format(name, normal_label),
         xlabel='Log2(Mean)', ylabel='Log2(FC)',
         color='colorby',
         marker='colorby',
         tooltips=tooltips,
```

```
          legend=True,
          palette=Category10_4,
          height=500,
          width=500)

p_neg = Scatter(neg_plot, x='mean', y='l2fc',
          title='{} Random Sample'.format(name),
          xlabel='Log2(Mean)', ylabel='Log2(FC)',
          color='colorby',
          marker='colorby',
          tooltips=tooltips,
          legend=True,
          palette=Category10_4,
          height=500,
          width=500)

p_reg.title.align = 'center'
p_neg.title.align = 'center'

p_reg.legend.location = 'bottom_right'
p_neg.legend.location = 'bottom_right'


p_reg.y_range = Range1d(start=-11, end=11)
p_neg.y_range = Range1d(start=-11, end=11)

p_reg.x_range = Range1d(start=0, end=20)
p_neg.x_range = Range1d(start=0, end=20)

try:
os.mkdir('negative_control')
except:
pass

#show(p_reg)
#show(p_neg)

reg = os.path.join('negative_control',
            name + '-{}.html'.format(comparison))
neg = os.path.join('negative_control',
            name + '-{}-negative.html'.format(comparison))

save(p_reg, reg)
save(p_neg, neg)

# reset_output()
```

```
In [71]: MA_plots(df, tissues, metadata, comparison='NvT')

Kidney
Number of samples        t: 882        n: 129
Creating Dataframe
Labeling Samples
Building Plots
```

```
In [75]: MA_plots(df, tissues, metadata, comparison='GvT')

Adrenal
Number of samples        t: 254        n: 125
```

```
Creating Dataframe
Labeling Samples
Building Plots
Brain
Number of samples        t: 662        n: 1142
Creating Dataframe
Labeling Samples
Building Plots
Ovary
Number of samples        t: 420        n: 88
Creating Dataframe
Labeling Samples
Building Plots
Pancreas
Number of samples        t: 178        n: 165
Creating Dataframe
Labeling Samples
Building Plots
Testis
Number of samples        t: 148        n: 165
Creating Dataframe
Labeling Samples
Building Plots
```

```
In [73]: MA_plots(df, tissues, metadata, comparison='NvT', matched=True)
```

```
Kidney
Number of samples        t: 129        n: 129
Creating Dataframe
Labeling Samples
Building Plots
```

5.6.3 Kidney Spearman Correlation for Log2FC

```
In [ ]: tissue_correlations = {}
for tissue in tissues:
name = '-'.join(tissue)
print name
samples = get_samples_for_tissue(metadata, tissue)
samples = [x for x in samples if x in df.columns]
g = list(set([x for x in samples if not x.startswith('TCGA')]))
t = list(set([x for x in samples if x.startswith('TCGA')
  and x.endswith('01')]))
n = list(set([x for x in samples if x.startswith('TCGA')
  and x.endswith('11')]))

if g and t and n:
g_med = df[g].T.median(axis=0)
t_med = df[t].T.median(axis=0)
n_med = df[n].T.median(axis=0)

nt_l2fc = get_l2fc(n_med, t_med)
gt_l2fc = get_l2fc(g_med, t_med)

plot = pd.DataFrame()
plot['NvT'] = nt_l2fc
plot['GvT'] = gt_l2fc
plot.index = nt_l2fc.index
```

```
tissue_correlations[name] = plot
else:
print '\tNot enough samples to make a full comparison'
```

```
In [ ]: f, ax = plt.subplots(4, 4, sharex=True, sharey=True, figsize=(12, 12))
ax = ax.flatten()
for i, tissue in enumerate(sorted(tissue_correlations.keys())):
plot = tissue_correlations[tissue]
sns.regplot(data=plot, x='NvT', y='GvT', scatter_kws={'alpha': 0.3},
ax=ax[i])
ax[i].set_title(tissue)
```

```
In [ ]: for tissue in sorted(tissue_correlations.keys()):
plot = tissue_correlations[tissue]
sns.jointplot(data=plot, kind='reg', x='NvT', y='GvT',
  scatter_kws={'alpha':0.3})
plt.xlabel(tissue + ' Log2 Fold Change Normal v. Tumor')
plt.ylabel(tissue + ' Log2 Fold Change GTEx v. Tumor');
```

5.6.4 MA Plot for Kidney

```
In [36]: p = figure(plot_width=400, plot_height=400)
p.square([1, 2, 3, 4, 5], [6, 7, 2, 4, 5],
size=20, color="olive", alpha=0.5)
p.circle([2, 3, 4, 5, 6], [6, 7, 2, 4, 5],
size=20, color="olive", alpha=0.5)
show(p)
```

Make sample hash map

```
In [ ]: samples = {}
for tissue in tissues:
name = '-'.join(tissue)
samps = get_samples_for_tissue(metadata, tissue)
samps = [x for x in samps if x in df.columns]
samples[name] = samps
```

```
In [88]: with open('samples.pickle', 'w') as f:
pickle.dump(samples, f)
```

6 Supplementary Figures

6.1 Cost per Genome

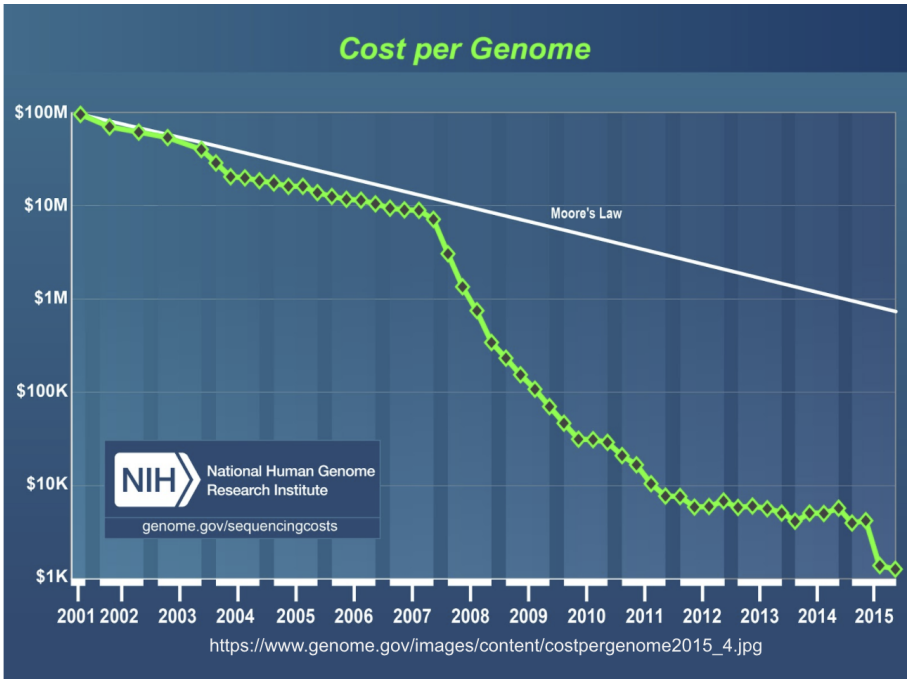


Figure 1: Cost to sequence a human genome since the Human Genome Project

6.2 Spark Cluster using Toil Services

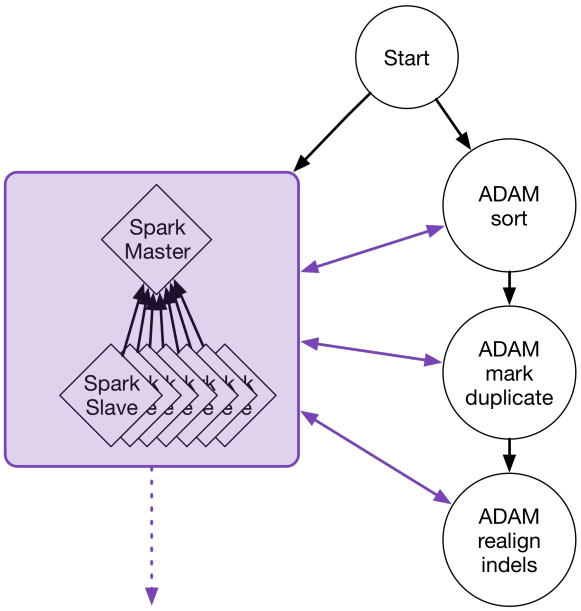


Figure 1: Running ADAM, a genomics processing workflow, which interacts with a SPARK cluster. Toil runs the SPARK cluster as a service job which is maintained for the duration of the workflow.

6.3 Expression Concordance Modified by CutAdapt

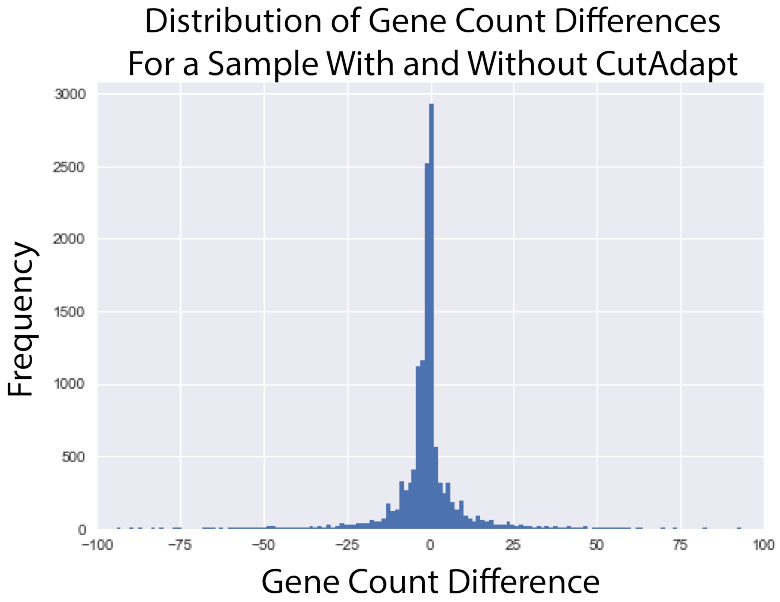


Figure 2: Expression differences from a sample obtained by running the toil-rnaseq workflow with and without CutAdapt. A total of 13,559 non-zero count differences were detected, with the largest differences being several thousand counts.

6.4 Large-scale Compute Metrics

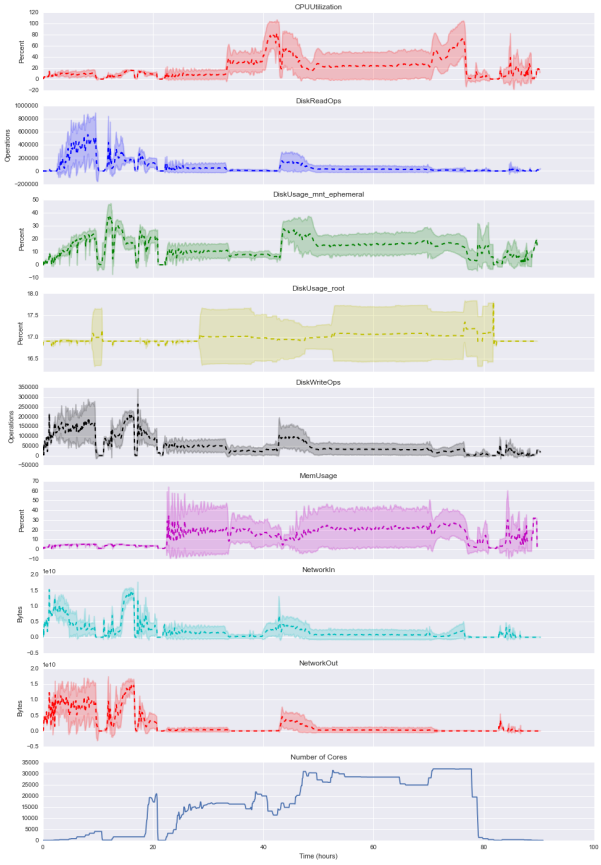


Figure 3: Aggregate metrics collected during the Toil RNA-seq large-scale compute. Metrics were collected every 5 minutes and averaged across every instance running at that time point. Complete termination of the Mesos framework occurred around hours 10 and 20, from which the workflow was resumed.

6.5 Pairwise DESeq2 Runtime

Figure 4: Runtimes of pairwise DESeq2 between 112 normal and 1,108 tumor breast cancer samples from TCGA over a range of partition sizes for the tumor group.

6.6 Expression Distributions for CD70 and CA9

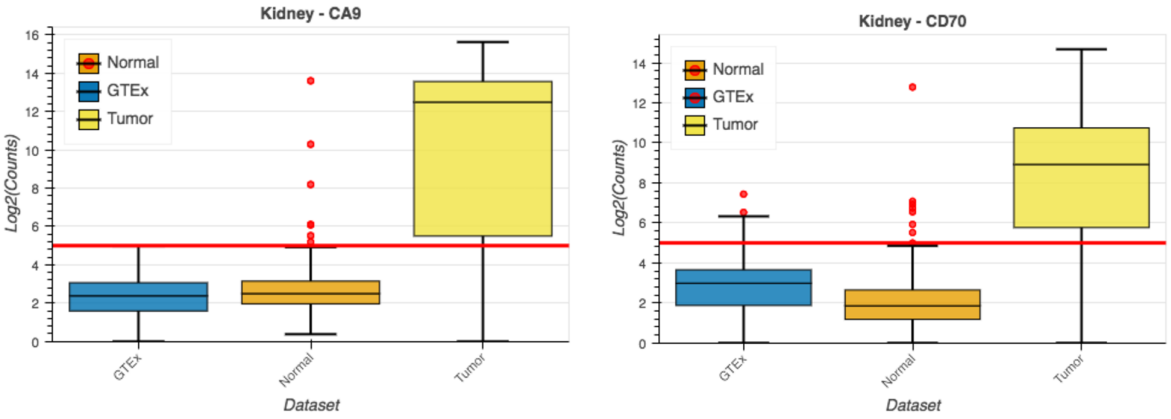


Figure 5: Gene Expression Distributions for CD70 and CA9 across GTEx and TCGA. The pattern of low expression in normal tissue and high expression in tumor tissue indicate possible targets for monoclonal antibodies if the gene encodes a protein expressed on the surface of the cell.

7 Supplementary Notes

7.1 Static Job Example

```
from toil.job import Job

def helloWorld(job, message, memory="2G", cores=2, disk="3G"):
    job.fileStore.logToMaster("Hello world, "
    "I have a message: %s" % message) # This uses a logging function
    # of the toil.fileStore.FileStore class

j1 = Job.wrapJobFn(helloWorld, "first")
j2 = j1.addChildJobFn(helloWorld, "second or third")
j3 = j1.addChildJobFn(helloWorld, "second or third")
j4 = j1.addFollowOnJobFn(helloWorld, "last")

if __name__=="__main__":
    options = Job.Runner.getDefaultOptions("./toilWorkflowRun")
    options.logLevel = "INFO"
    Job.Runner.startToil(j1, options)
```

7.2 Dynamic Job Creation

```
from toil.job import Job

def binaryStringFn(job, depth, message=""):
    if depth > 0:
        job.addChildJobFn(binaryStringFn, depth-1, message + "0")
        job.addChildJobFn(binaryStringFn, depth-1, message + "1")
    else:
        job.fileStore.logToMaster("Binary string: %s" % message)

if __name__=="__main__":
    options = Job.Runner.getDefaultOptions("./toilWorkflowRun")
    options.logLevel = "INFO"
    Job.Runner.startToil(Job.wrapJobFn(binaryStringFn, depth=5), options)
```

7.3 Promises

```
from toil.job import Job

def fn(job, i):
    job.fileStore.logToMaster("i is: %s" % i, level=100)
    return i+1

j1 = Job.wrapJobFn(fn, 1)
j2 = j1.addChildJobFn(fn, j1.rv())
j3 = j1.addFollowOnJobFn(fn, j2.rv())

if __name__=="__main__":
    options = Job.Runner.getDefaultOptions("./toilWorkflowRun")
    options.logLevel = "INFO"
    Job.Runner.startToil(j1, options)
```

References

[1] Zhong Wang, Mark Gerstein, and Michael Snyder. RNA-Seq: A revolutionary tool for transcriptomics. Nature Reviews Genetics, 10(1):57–63, January 2009.

[2] The Cancer Genome Atlas Research Network, John N. Weinstein, Eric A. Collisson, Gordon B. Mills, Kenna R. Mills Shaw, Brad A. Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, and Joshua M. Stuart. The Cancer Genome Atlas Pan-Cancer analysis project. Nature Genetics, 45(10):1113–1120, October 2013.

[3] Toil — Toil 3.6.1a1 documentation. https://toil.readthedocs.io/en/releases-3.6.x/.

[4] Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain. Makeflow: A portable abstraction for data intensive computing on clusters, clouds, and grids. In Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, page 1. ACM, 2012.

[5] Jeremy Goecks, Anton Nekrutenko, and James Taylor. Galaxy: A comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome biology, 11(8):R86, 2010.

[6] Brian D O'Connor, Barry Merriman, and Stanley F. Nelson. SeqWare Query Engine: Storing and searching sequence data in the cloud. BMC bioinformatics, 11(12):S2, 2010.

[7] Erik Bernhardsson and Elias Freider. Luigi.

[8] GTEx Consortium and others. The Genotype-Tissue Expression (GTEx) pilot analysis: Multitissue gene regulation in humans. Science, 348(6235):648–660, 2015.

[9] GenomeOC. Therapeutically Applicable Research to Generate Effective Treatments. https://ocg.cancer.gov/programs/target, 2013-01-18T15:44-05:00.

[10] Pacific Pediatric Neuro-Oncology Consortium. http://www.pnoc.us/.

[11] Docker. https://www.docker.com/.

[12] Docker Hub. https://hub.docker.com/.

[13] Quay Container Registry · Quay. https://quay.io/.

[14] Ryan Chamberlain and Jennifer Schommer. Using Docker to support reproducible research. DOI: http://dx. doi. org/10.6084/m9. figshare, 1101910, 2014.

[15] Index of /public/mRNAseq_TCGA. https://webshare.bioinf.unc.edu/public/mRNAseq_TCGA/.

[16] BD2KGenomics/toil-rnaseq. https://github.com/BD2KGenomics/toil-rnaseq.

[17] A. Dobin, C. A. Davis, F. Schlesinger, J. Drenkow, C. Zaleski, S. Jha, P. Batut, M. Chaisson, and T. R. Gingeras. STAR: Ultrafast universal RNA-seq aligner. Bioinformatics, 29(1):15–21, January 2013.

[18] Simon. Andrews. FastQC: A quality control tool for high throughput sequence data., 2010.

[19] Marcel Martin. Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet.journal, 17(1):pp. 10–12, May 2011.

[20] Mary Goldman, Brian Craft, Teresa Swatloski, Melissa Cline, Olena Morozova, Mark Diekhans, David Haussler, and Jingchun Zhu. The UCSC Cancer Genomics Browser: Update 2015. Nucleic Acids Research, 43(D1):D812–D817, January 2015.

[21] John Vivian, Arjun Rao, Frank Austin Nothaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D. Deran, Audrey Musselman-Brown, and others. Rapid and efficient analysis of 20,000 RNA-seq samples with Toil. bioRxiv, page 062497, 2016.

[22] John Vivian, Arjun Arkal Rao, Frank Austin Nothaft, Christopher Ketchum, Joel Armstrong, Adam Novak, Jacob Pfeil, Jake Narkizian, Alden D. Deran, Audrey Musselman-Brown, Hannes Schmidt, Peter Amstutz, Brian Craft, Mary Goldman, Kate Rosenbloom, Melissa Cline, Brian O'Connor, Megan Hanna, Chet Birger, W. James Kent, David A. Patterson, Anthony D. Joseph, Jingchun Zhu, Sasha Zaranek, Gad Getz, David Haussler, and Benedict Paten. Toil enables reproducible, open source, big biomedical data analyses. Nature Biotechnology, 35(4):314–316, April 2017.

[23] Human Cell Atlas. https://www.humancellatlas.org/.

[24] Nicolas L. Bray, Harold Pimentel, Páll Melsted, and Lior Pachter. Near-optimal probabilistic RNA-seq quantification. Nature Biotechnology, 34(5):525–527, May 2016.

[25] Mark D. Robinson and Alicia Oshlack. A scaling normalization method for differential expression analysis of RNA-seq data. Genome biology, 11(3):R25, 2010.

[26] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. Genome biology, 11(10):R106, 2010.

[27] Michael I Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12), December 2014.

[28] Jun Li. PoissonSeq, 2012.

[29] Marie-Agnès Dillies, Andrea Rau, Julie Aubert, Christelle Hennequet-Antier, Marine Jeanmougin, Nicolas Servant, Céline Keime, Guillemette Marot, David Castel, Jordi Estelle, Gregory Guernec, Bernd Jagla, Luc Jouneau, Denis Laloë, Caroline Le Gall, Brigitte Schaëffer, Stéphane Le Crom, Mickaël Guedj, and Florence Jaffrézic. A comprehensive evaluation of normalization methods for Illumina high-throughput RNA sequencing data analysis. Briefings in Bioinformatics, 14(6):671–683, November 2013.

[30] Yan Guo, Quanhu Sheng, Jiang Li, Fei Ye, David C. Samuels, and Yu Shyr. Large Scale Comparison of Gene Expression Levels by Microarrays and RNAseq Using TCGA Data. PLoS ONE, 8(8):e71462, August 2013.

[31] Li Peng, Xiu Wu Bian, Di Kang Li, Chuan Xu, Guang Ming Wang, Qing You Xia, and Qing Xiong. Large-scale RNA-Seq Transcriptome Analysis of 4043 Cancers and 548 Normal Tissue Controls across 12 TCGA Cancer Types. Scientific Reports, 5:13413, August 2015.

[32] Ehsan Amid, Nikos Vlassis, and Manfred K. Warmuth. Low-dimensional Data Embedding via Robust Ranking. arXiv:1611.09957 [cs, stat], November 2016.

[33] Laurens van der Maaten and Geoffrey Hinton. Visualizing Data using t-SNE. Journal of Machine Learning Research, 9(Nov):2579–2605, 2008.

[34] Bo Li and Colin N. Dewey. RSEM: Accurate transcript quantification from RNA-Seq data with or without a reference genome. BMC bioinformatics, 12(1):323, 2011.

[35] D. G. Altman and J. M. Bland. Measurement in Medicine: The Analysis of Method Comparison Studies. Journal of the Royal Statistical Society. Series D (The Statistician), 32(3):307–317, 1983.

[36] Hirotsugu Uemura, Kiyohide Fujimoto, Motoyoshi Tanaka, Motoyoshi Yoshikawa, Yoshihiko Hirao, Shigeya Uejima, Kazuhiro Yoshikawa, and Kyogo Itoh. A Phase I Trial of Vaccination of CA9-Derived Peptides for HLA-A24-Positive Patients with Cytokine-Refractory Metastatic Renal Cell Carcinoma. Clinical Cancer Research, 12(6):1768–1775, March 2006.

[37] Iqbal S. Grewal. CD70 as a therapeutic target in human malignancies. Expert Opinion on Therapeutic Targets, 12(3):341–351, March 2008.

[38] Dario Neri and Claudiu T. Supuran. Interfering with pH regulation in tumours as a therapeutic strategy. Nature Reviews Drug Discovery, 10(10):767–777, October 2011.

[39] Jens Bedke and Arnulf Stenzl. Immunotherapeutic strategies for the treatment of renal cell carcinoma: Where are we now? Expert Review of Anticancer Therapy, 13(12):1399–1408, December 2013.

[40] Damaris Bausch-Fluck, Andreas Hofmann, Thomas Bock, Andreas P. Frei, Ferdinando Cerciello, Andrea Jacobs, Hansjoerg Moest, Ulrich Omasits, Rebekah L. Gundry, Charles Yoon, Ralph Schiess, Alexander Schmidt, Paulina Mirkowska, Anetta Härtlová, Jennifer E. Van Eyk, Jean-Pierre Bourquin, Ruedi Aebersold, Kenneth R. Boheler, Peter Zandstra, and Bernd Wollscheid. A Mass Spectrometric-Derived Cell Surface Protein Atlas. PLOS ONE, 10(4):e0121314, April 2015.

[41] Wilson Wen Bin Goh, Wei Wang, and Limsoon Wong. Why Batch Effects Matter in Omics Data, and How to Avoid Them. Trends in Biotechnology, 35(6):498–507, June 2017.

[42] J Luo, M Schumacher, A Scherer, D Sanoudou, D Megherbi, T Davison, T Shi, W Tong, L Shi, H Hong, C Zhao, F Elloumi, W Shi, R Thomas, S Lin, G Tillinghast, G Liu, Y Zhou, D Herman, Y Li, Y Deng, H Fang, P Bushel, M Woods, and J Zhang. A comparison of batch effect removal methods for enhancement of prediction performance using MAQC-II microarray gene expression data. The Pharmacogenomics Journal, 10(4):278–291, August 2010.

[43] Tse-Wen Chang. Binding of cells to matrixes of distinct antibodies coated on solid surface. Journal of Immunological Methods, 65(1):217–223, December 1983.

[44] Monica Benito, Joel Parker, Quan Du, Junyuan Wu, Dong Xiang, Charles M. Perou, and J. S. Marron. Adjustment of systematic microarray data biases. Bioinformatics, 20(1):105–114, January 2004.

[45] Andrew H. Sims, Graeme J. Smethurst, Yvonne Hey, Michal J. Okoniewski, Stuart D. Pepper, Anthony Howell, Crispin J. Miller, and Robert B. Clarke. The removal of multiplicative, systematic bias allows integration of breast cancer gene expression datasets – improving meta-analysis and prediction of prognosis. BMC Medical Genomics, 1(1):42, September 2008.

[46] Jeffrey leek and John Storey. Capturing Heterogeneity in Gene Expression Studies by Surrogate Variable Analysis. http://journals.plos.org/plosgenetics/article?id=10.1371/journal.pgen.0030161.

[47] W. Evan Johnson, Cheng Li, and Ariel Rabinovic. Adjusting batch effects in microarray expression data using empirical Bayes methods. Biostatistics, 8(1):118–127, January 2007.

[48] Chao Chen, Kay Grennan, Judith Badner, Dandan Zhang, Elliot Gershon, Li Jin, and Chunyu Liu. Removing Batch Effects in Analysis of Expression Microarray Data: An Evaluation of Six Batch Adjustment Methods. PLOS ONE, 6(2):e17238, February 2011.

[49] Davide Risso, John Ngai, Terence P. Speed, and Sandrine Dudoit. Normalization of RNA-seq data using factor analysis of control genes or samples. Nature Biotechnology, 32(9):896–902, September 2014.

[50] Christian Müller, Arne Schillert, Caroline Röthemeier, David-Alexandre Trégouët, Carole Proust, Harald Binder, Norbert Pfeiffer, Manfred Beutel, Karl J. Lackner, Renate B. Schnabel, Laurence Tiret, Philipp S. Wild, Stefan Blankenberg, Tanja Zeller, and Andreas Ziegler. Removing Batch Effects from Longitudinal Gene Expression - Quantile Normalization Plus ComBat as Best Approach for Microarray Transcriptome Data. PLOS ONE, 11(6):e0156594, June 2016.

[51] Uri Shaham, Kelly P. Stanton, Jun Zhao, Huamin Li, Khadir Raddassi, Ruth Montgomery, and Yuval Kluger. Removal of batch effects using distribution-matching residual networks. Bioinformatics, 33(16):2539–2546, August 2017.

[52] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity Mappings in Deep Residual Networks. In Computer Vision – ECCV 2016, Lecture Notes in Computer Science, pages 630–645. Springer, Cham, October 2016.

[53] Karsten M. Borgwardt, Arthur Gretton, Malte J. Rasch, Hans-Peter Kriegel, Bernhard Schölkopf, and Alex J. Smola. Integrating structured biological data by Kernel Maximum Mean Discrepancy. Bioinformatics, 22(14):e49–e57, July 2006.

[54] Yee W. Teh, Michael I. Jordan, Matthew J. Beal, and David M. Blei. Sharing clusters among related groups: Hierarchical Dirichlet processes. In Advances in Neural Information Processing Systems, pages 1385–1392, 2005.

[55] Arthur C. Rand, Miten Jain, Jordan M. Eizenga, Audrey Musselman-Brown, Hugh E. Olsen, Mark Akeson, and Benedict Paten. Mapping DNA methylation with high-throughput nanopore sequencing. Nature Methods, 14(4):411–413, April 2017.

[56] Jordan Eizenga. Hdp_mixture: Hierarchical Dirichlet process mixture model, January 2016.

[57] Andy B. Yoo, Morris A. Jette, and Mark Grondona. SLURM: Simple Linux Utility for Resource Management. In SpringerLink, pages 44–60. Springer, Berlin, Heidelberg, June 2003.

[58] Wolfgang Gentzsch. Sun grid engine: Towards creating a compute power grid. In Cluster Computing and the Grid, 2001. Proceedings. First IEEE/ACM International Symposium On, pages 35–36. IEEE, 2001.

[59] Benjamin Hindman, Andy Konwinski, Matei Zaharia, Ali Ghodsi, Anthony D. Joseph, Randy H. Katz, Scott Shenker, and Ion Stoica. Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center. In NSDI, volume 11, pages 22–22, 2011.

[60] Jim Kent — Center for Biomolecular Science and Engineering. https://cbse.soe.ucsc.edu/people/kent.

[61] Claire R. Williams, Alyssa Baccarella, Jay Z. Parrish, and Charles C. Kim. Trimming of sequence reads alters RNA-Seq gene expression estimates. BMC Bioinformatics, 17:103, 2016.

[62] What the FPKM? A review of RNA-Seq expression units, 2014-05-08T18:55:06+00:00.

[63] K. Wang, D. Singh, Z. Zeng, S. J. Coleman, Y. Huang, G. L. Savich, X. He, P. Mieczkowski, S. A. Grimm, C. M. Perou, J. N. MacLeod, D. Y. Chiang, J. F. Prins, and J. Liu. MapSplice: Accurate mapping of RNA-seq reads for splice junction discovery. Nucleic Acids Research, 38(18):e178–e178, October 2010.

[64] Dockstore. https://dockstore.org/.

[65] Blue Collar Bioinformatics. http://bcb.io/.

[66] Analysis Core. http://ucsc-cgl.org/.

[67] Children's Hospital of Philadelphia. Children's Hospital of Philadelphia. http://www.chop.edu.

[68] UW Genome Sciences. http://www.gs.washington.edu/.

[69] ucsf_admin. California Kids Cancer Comparison. http://www.ciapm.org/project/california-kids-cancer-comparison, 2015-09-30T10:48:23-07:00.

[70] Stand Up To Cancer — SU2C-PCF2 Dream Team: Targeting Adaptive Pathways in Metastatic Treatment-Resistant Prostate Cancer. https://www.standup2cancer.org/dream_teams/view/targeting_adaptive_pathways_in_metastatic_crpc.

[71] Canada's Michael Smith Genome Sciences Centre. http://www.bcgsc.ca/.

[72] Treehouse Childhood Cancer Initiative – UC Santa Cruz Genomics Institute. https://treehousegenomics.soe.ucsc.edu/.

[73] Broad Institute. https://www.broadinstitute.org/.

[74] Olena Morozova, Yulia Newton, Avanthi Tayi Shah, Holly Beale, Du Linh Lam, John Vivian, Isabel Bjork, Theodore Goldstein, Josh Stuart, Sofie Salama, E. Alejandro Sweet-Cordero, and David Haussler. Abstract 4890: A pan-cancer analysis framework for incorporating gene expression information into clinical interpretation of pediatric cancer genomic data. Cancer Research, 77(13 Supplement):4890–4890, July 2017.

[75] BD2KGenomics/toil-scripts. https://github.com/BD2KGenomics/toil-scripts.

[76] Cgcloud: Image and VM management for Jenkins, Spark and Mesos clusters in EC2, March 2017.