Parallelizing GENSENG

Comprehensive Examination for M.S. Computer Science

April 22, 2014

Benjamin Tzou

Abstract

Structural variations are important contributors to genomic variation. Copy number variations are a particular form of structural variation that insert or delete copies of segments in the genome. Recent research has shown that these CNVs can have serious medical implications and has been associated with conditions ranging from neurological disorders such as schizophrenia to immune dysfunction in cancer and HIV infection. With the advent of high throughput sequencing, CNV detection is poised for rapid development. GENSENG is one method for detecting CNVs that has shown promise, demonstrating accuracy that rivals the state of the art, but requires a comparatively long time to run. This project explores multi-core parallelization to reduce the run-time of GENSENG. The resulting parallel implementation is 3.7 times as fast on an 8-core machine while maintaining accuracy and runs without issue on single and multi-core machines.

Parallelizing GENSENG

Comprehensive Examination for M.S. Computer Science

April 22, 2014

Benjamin Tzou

1. Introduction

Structural variation refers to inversions, translocations, insertions and deletions of large genomic regions within the chromosomes of an organism. Because insertions and deletions can alter the number of copies of a gene within the genome, they are termed copy number variations (CNVs). These account for roughly 12% of the human genome (Stankiewicz 2010) and have been associated with schizophrenia, autism, immune dysfunction and various forms of cancer (Cook 2008, St Clair D 2008, Cappuzzo 2005). Copy number differences are also a source of error and must be corrected for in functional genomic studies (Laird 2010, Rashid 2011).

CNV detection stands to benefit from the rapid development of high throughput sequencing technologies but challenges of speed and accuracy persist. GENSENG is one algorithm for detecting CNVs that uses read depth information and has demonstrated accuracy that rivals or exceeds current, widely used algorithms, such as CNVnator (Szatkiewicz 2013, Abyzov 2011). GENSENG, however, requires protracted run-times on the order of tens of hours for the human genome and is unlikely to scale for batch analyses. With the aim of reducing run-time, this paper explores multi-core parallelization of GENSENG while otherwise maintaining accuracy.

2. CNV detection

CNVs are considered unbalanced genomic alterations, which result in the gain or loss of genetic material through duplication and deletion respectively. Duplications and deletions can affect multiple copies of a gene at once, such as when two of three copies of a gene are deleted at once. CNVs are significant and worthwhile targets of analysis due to their association with a wide range of medical conditions, from neurological disorders such as schizophrenia to immunological disorders such as lupus and susceptibility to HIV (Aitman 2006, Gonzalez 2005).

CNV detection methods are generally either microarray-based or sequencing-based. Microarray-based technologies included array comparative genomic hybridization and virtual karyotyping with SNP microarrays (Iafrate 2004, Sebat 2004). Sequencing-based technologies take advantage of next-generation sequencing and promise high resolution and throughput. These technologies further consist of four approaches, including read-pair, split read, assembly and read depth, with each varying in specificity, sensitivity and appropriateness given the sample data. The read depth approach intuitively looks for lower or higher than expected sequencing coverage, which suggest deletions or insertions. For example, a genomic region in the sample that has twice the expected sequencing coverage is a possible duplication event since duplicated regions will on average generate twice the number of reads for the original sequence.

3. GENSENG

While read depth approaches are intuitive, care must be taken to validate model assumptions and to eliminate subtle biases introduced by the sequencing and mapping processes. GENSENG specifically and directly handles biases in sequence mapping associated with GC content and the inherent mappability of a region. GENSENG also examines and corrects for the common, flawed assumption that read depth follows a Poisson distribution. With a focus on bias correction, GENSENG introduces a novel read depth-based method for detecting CNVs that utilizes a hidden Markov model (HMM) and negative binomial regression framework. While HMMs often model a sequence of states varying over time, GENSENG models copy number state from one window to the next within a genetic sequence. Windows, in the context of GENSENG, contain a constant, user-specified number of bases and are non-overlapping (or overlapping in the case of autoregression).

Various biases are possible when detecting CNVs from read depth data. With regards to GC content, the authors related the proportion of G and C bases to the read depth per genomic region and confirmed that sequences with low and high GC content tend to have lower read depth (Figure 1a). Without identifying and correcting for this bias, low and high GC content regions can erroneously be called as deletions. With mappability, the authors observed that read depth varies with the uniqueness of the underlying sequence (Figure 1b). Intuitively, genomic regions with repetitive sequences map poorly onto themselves and result in low read depth coverage; a low mappability score can help control for the low read depth. Finally, authors observed that a Poisson distribution often does not adequately model the variation in read depth (Figure 2). In order to accommodate the overdispersion, the authors augmented their Poisson model to take an additional parameter, resulting in the negative binomial distribution.

Figure 1. The relationship between read depth and GC content is shown in Figure 1a. The relationship between read depth and mappability is shown in Figure 1b.

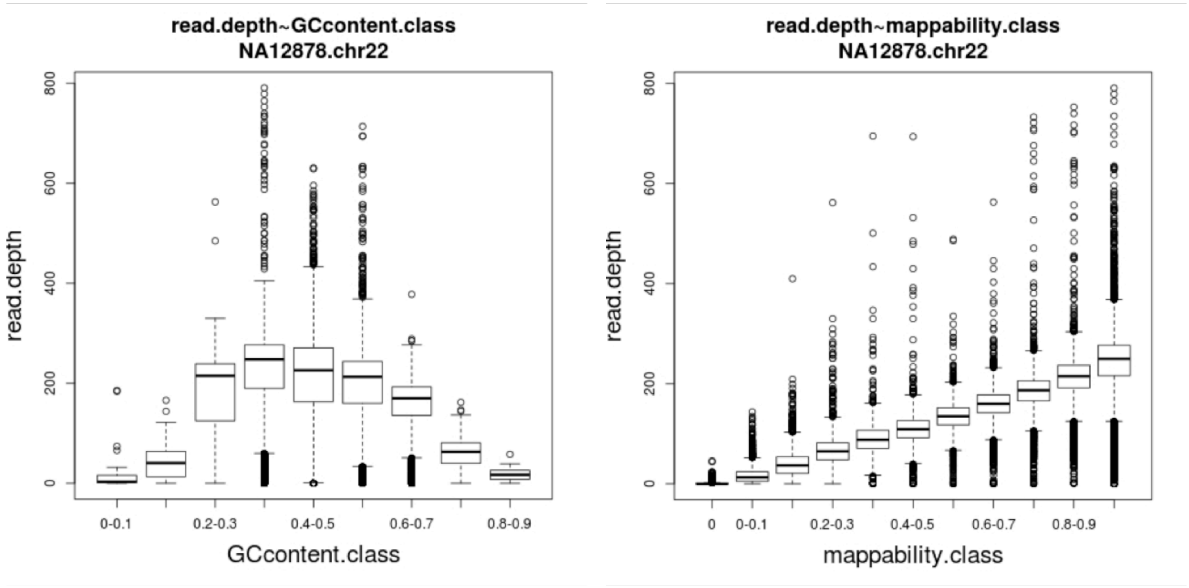
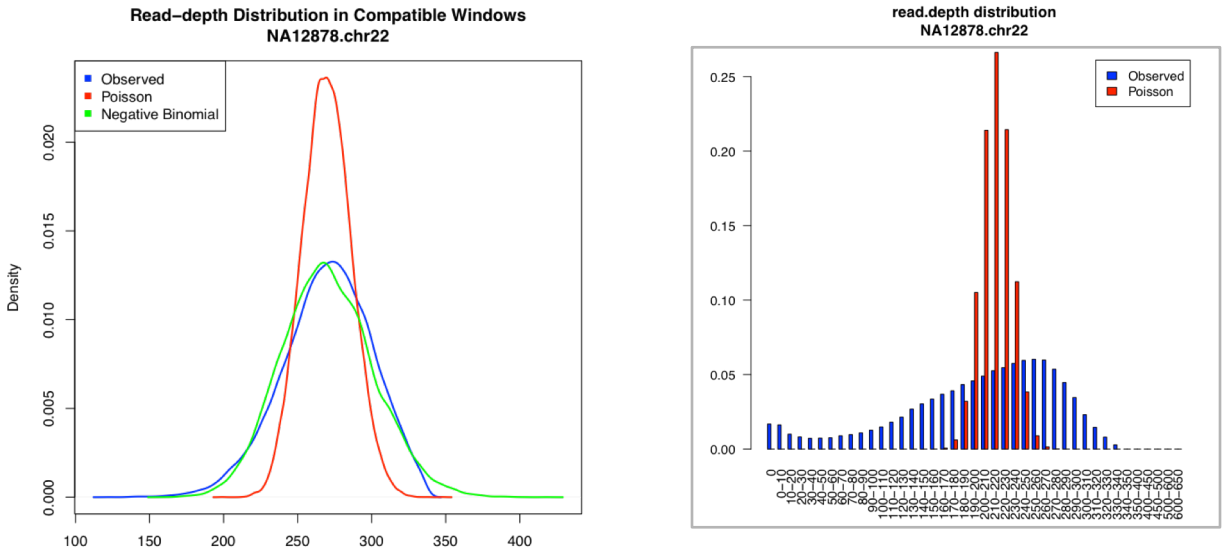


Figure 2. Modeling read depth with the Poisson and negative binomial distributions



4. Parallelizing GENSENG

4.1 Overview

While GENSENG outperforms competing algorithms such as CNVnator in accuracy, the current, sequential implementation requires hours to complete when analyzing an entire human genome. For single samples, the large time requirement may be acceptable if a sample needs to be run only once. However, when running many samples, the time required is prohibitive and precludes large-scale implementation. This project explores the use of parallelization to decrease the run-time of GENSENG.

By definition, an HMM models sequential information and GENSENG not constitute an exception. Given this structural aspect of HMMs, there currently exist no straightforward, significant means to train or perform inference on an HMM model in parallel. Similarly, solving negative binomial regression within GENSENG involves iterative procedures for finding convergent model parameters, which by definition are also sequential. Still, close inspection reveals potential areas for parallelization within the sequential steps of regression and HMM training. Since the original code base is written in C++, this project uses OpenMP, a set of compiler directives for C++, to produce parallel code. Programs compiled with OpenMP directives run on both single core and multi-core processors with no user intervention.

HMM training involves the Baum-Welch algorithm, which uses the forward-backward and expectation-maximization algorithms to find the minimum likelihood estimates (MLEs) of the HMM parameters. From another perspective, the algorithmic flow of GENSENG can be viewed as consisting of two major steps: inference and re-estimation. Inference includes the forward and backward algorithms, with each having potential for parallelization generally for each state during each sequential step, and the calculation for posterior probability through the forward and backward probabilities. Re-estimation includes setting

new values for the initial state, transition and emission probabilities, which involves the regression model fitting. The sections below detail how parallelization was achieved for each step.

For notation, we use:

q_i: the state at position i

π_z: the probability that the state at position 0 is state z

$e(i, z)$: the emission probability of state z at position i α

$\alpha_i(j, k)$: the transition probability from state j to k from state i-1 to i

4.2 Inference

The inference portion of the GENSENG algorithm calculates the forward probabilities, backward probabilities, likelihood and posterior probabilities. The calculations for the forward, backward and posterior probabilities were parallelized but the calculation for likelihood was not.

The forward algorithm for HMMs involves calculating the probability of a state at a certain time given the history of evidence. Using the above notation, we have the following equations for the base case and recursive case for $i \in (2: L)$ and for $z \in (1: N)$:

$$f(1, z) = \pi_z e(1, z)$$

$$f(1, z) = e(1, z) \sum_j f(i-1, j)\alpha_i(j, z)$$

The recursive calculation for f at i depends on the value of f at position i-1, which is sequential and not parallelizable. However, it is possible to parallelize on the calculation of f for each state at every i since dependencies exist between but not within positions. GENSENG is configured to distinguish only a small number of states so the benefit from parallelization is likely to be small due to the overhead of coordinating threads. The original code appears below, followed by the parallelized version. Note that, in addition to the compiler directive `#pragma omp parallel for`, it is necessary to allocate a new array for each state since threads require their own memory space for performing the calculation correctly. With the new code, sequential execution is slightly less efficient, but the alternative is to allocate memory for the maximum number of threads at once and calculate indices for each thread, which provides an arguable gain in efficiency at the cost of significantly less readable code. Sample code for this computation is provided below.

Code 1. Sequential and parallel

```
void HMModel::computAlpha(void)   // sequential
{
  ...

  double *v = new double[nSTATES];
  for(int i = 1; i < nLength; ++i)
  {
    for(int j = 0; j < nSTATES; ++j)
    {
```

```
      for(int k = 0; k < nSTATES; ++k)
      {
        v[k] = pAlpha[i-1][k]+log(pTran[i][k][j]);
      }
      pAlpha[i][j] = MathTools::logsumexp(v, nSTATES)+log(pEmissTbl[i][j]);
    }
  }
  delete []v;
}

void HMModel::computAlpha(void)   // parallel
{
  for(int i = 0; i < nSTATES; ++i)
  {
    pAlpha[0][i] = log(pPi[i]) + log(pEmissTbl[0][i]);
  }

  for(int i = 1; i < nLength; ++i)
  {
    #pragma omp parallel for
    for(int j = 0; j < nSTATES; ++j)
    {
      double *v = new double[nSTATES];
      for(int k = 0; k < nSTATES; ++k)
      {
        v[k] = pAlpha[i-1][k]+log(pTran[i][k][j]);
      }
      pAlpha[i][j] = MathTools::logsumexp(v, nSTATES)+log(pEmissTbl[i][j]);
      delete []v;
    }
  }
}
```

Likelihood can be calculated upon completing the forward algorithm. Since it involves summing the probability of each state in the last position, parallelization can be done but comparison reveals that it in fact increases execution time. With the small state space in GENSENG, the speedup offered by parallelizing this summation does not overcome the costs of thread management. The equation below may provide intuition:

$$p(\mathbf{X}|\Theta) = \sum_{z} f(L, z)$$

The backward algorithm, which gives the probability of the ending observations given state *z* at position *i*, is parallelized in similar fashion to the forward algorithm. Dependencies exist between positions but parallelization is possible for each state at a given position. This follows simply from the backward

algorithm equation, where the base case handles the last position and the recursive case handles $i \in (L\colon 2)$ and $z \in (1\colon N)$.

$$b(L,z) = 1$$

$$b(i-1,z) = \sum_j [\alpha_i(z,j)e(i,j)b(i,j)]$$

The posterior probability can be calculated from the forward and backward probabilities for each state at each position. The nested for loop is easily parallelized since no dependencies exist between each posterior probability. This is evident in the equation:

$$\gamma(i,z) = \frac{f(i,z)b(i,z)}{\sum_z f(L,z)}$$

4.3 Re-estimation

Re-estimation in the context of GENSENG involves calculating new values for the initial state, transition and emission probabilities.

The initial state probabilities are calculated by multiplying the forward and backward probabilities together and dividing by the result by the likelihood. Since the number of probabilities to calculate depends on the number of states, which is small within GENSENG, parallelization was not performed for this calculation. The equation is given by:

$$\pi_z = \frac{f(1,z)b(1,z)}{p(\mathbf{X}|\Theta)}$$

Transition probabilities are calculated through a sequence of smaller calculations. Each transition probability describes the probability that a state in one position transitions to another state in the next position. As such, the transition matrix has length and width equal to the number of states. These probabilities can be calculated by using the previous value, following the below equation:

$$\bar{a}_{jk} = \frac{\sum_{i=2}^{L} f(i-1,j)a_{jk}\left(1-\exp\left(-\lambda_j d_i\right)\right)e(i,k)b(i,k)}{\sum_{i=2}^{L}\sum_{l\neq j} f(i-1,j)a_{jl}\left(1-\exp\left(-\lambda_j d_i\right)\right)e(i,l)b(i,l)}$$

To prevent overflow and underflow, the so-called log-sum-exp trick is used to provide an intermediate value called c_{jk} in the code.

$$\text{logsumexp}_j(v) = \log\left(\sum_j \exp\left(v_j\right)\right)$$

The value $\bar{a}_{jk}$ is calculated and combined with the log value of the previous transition probability to generate the new estimate using the below formulas:

$$c_{jk} = \text{logsumexp}_i\left[\log\left(f(i-1,j)\right) + \log\left(1-exp\left(-\lambda_j d_i\right)\right) + \log\left(e(i,k)\right) + \log\left(b(i,k)\right)\right]$$

$$\log\left(\bar{a}_{jk}\right) = \log\left(a_{jk}\right) + c_{jk} - \text{logsumexp}_{l\neq j}\left[\log\left(a_{jl}\right) + c_{jl}\right]$$

Notably, each value of c_{jk} requires a calculation that involves values from every position. The code therefore has a three-level nested-loop that parallelizes for a significant speedup.

Emission probabilities are also calculated through a multi-step process, which on the whole is sequential but with intermediate steps that are independent and highly parallelizable. This section will follow the notation and calculations described in the supplemental materials section of the original GENSENG paper. The emission probability is modeled as a mixture of the uniform distribution and the negative binomial distribution, which can be expressed as below. In the equation, c is the mixing probability, o_t is the read depth signal for window t, μ_{tj} is the mean read depth for window t given state j and ϕ_j is the overdispersion parameter given state j.

$$\frac{c}{R_m} + (1-c)\frac{\Gamma(o_t + \frac{1}{\phi_j})}{o_t!\,\Gamma(\frac{1}{\phi_j})}\left(\frac{1}{1+\phi_j\mu_{tj}}\right)^{1/\phi_j}\left(\frac{\phi_j\mu_{tj}}{1+\phi_j\mu_{tj}}\right)^{o_t}$$

To estimate the negative binomial parameters, GENSENG constructs a weighted generalized linear model function and alternatingly estimates ϕ and μ_{tj}, using one to calculate the other until convergence. First, ϕ is fixed and μ_{tj} is computed by fitting the weighted GLM using the iteratively reweighted least squares method (IRLS). Then μ_{tj} is fixed and ϕ is calculated using the Newton-Raphson method with weights.

Without delving into the details, the IRLS component involves multiple calculations over every window in the sample, which are independent and parallelizable. Within the IRLS code, the helper functions that perform these calculations are `wcenter`, `wssq`, and `wresid`, which calculate the mean, sum of squares and residuals, respectively, and these are each parallelized over all the windows.

Similarly, for the Newton-Raphson calculation for ϕ, the corresponding function in the code is `phi_ml`, which is dominated by computations in the helper function `score_info`, which provides a score and Fisher information on phi. `score_info` performs calculations over all windows and is similarly parallelized. `score_info` is in particular a good example of using special OpenMP features to obtain a speedup beyond normal thread parallelization. Because the loop within `score_info` sums over a for loop and essentially performs a reduction, OpenMP allows the use of the special directive `#pragma omp parallel for reduction(+:score1,info1)` to generate code that uses a register variable for the accumulation. A single local variable thus minimizes overhead in this special case.

5. Performance

On analyzing the sequential and parallel performance of GENSENG, it becomes clear there are several bottlenecks. The inference computations represent less than 5% of the time required for the re-estimation computations. GENSENG also writes to disk the results of its calculations at the end of each round of inference and re-estimation, which is more than a fifth of the total GENSENG run-time. Given additional time, it would be possible to re-write portions of GENSENG that reduce or eliminate the need for such expensive disk operations. Nonetheless, parallelization of GENSENG with minor code revisions resulted in a 3.7 times speedup when running GENSENG on an 8-core machine. The output of the parallelized implementation remains identical to the original sequential implementation.

*Chart 1. Execution time by code section before and after parallelization, averaged over 10 runs of 10 rounds. All times in seconds. * denotes a code segment that is, or contains segments that are, executed multiple times per training round.*

Code Section	sequential	parallel	speedup
Inference subsection			
forward probabilities	2.01	1.53	1.313
backward probabilities	3.07	1.9	1.613
likelihoods	0	0	--
posterior probabilities	0.01	0.01	1.019
Subsection time	5.09	3.45	1.473
Re-estimation subsection			
initial state probabilities	0	0	--
transition state probabilities			
cjks	1.84	0.44	4.146
fill_trans	0.04	0.04	1.164
Sub-subsection time	1.88	0.48	3.906
emission state probabilities			
mu_phi_xy	0.16	0.04	4.212
mu_phi_offset_init	0.14	0.03	5.564
mu_phi_offset_adjust	0.01	0.01	1.001
mu_phi_prior_load	0.09	0.02	4.627
mu_phi_prior_update	1.13	0.57	1.973
init_weights	0.04	0.02	2.567
IRLS iteration*	0.37	0.14	2.606
test_for_overdispersion	0.01	0	3.107
log_likelihood	0	0	--
phi_ml*	62.12	12.68	4.899
log_likelihood_nb	0.67	0.14	4.868
glm_nb_subtotal	95.88	21.99	4.359
mu_save_fitted	0.01	0.01	1.017
mu_phi_subtotal	157.53	34.79	4.528

fitting_glm_nb_subtotal	132.58	27.34	4.848
mu_adjust	0.01	0.01	1.046
mu_phi_ar_subtotal	197.61	41.11	4.806
fill_emission_table	1.14	0.58	1.975
Subsection time	359.73	78.33	4.592
Total time per round	364.82	81.76	4.462
Total execution time	3883.68	1048.11	3.705

From Chart 1, it is clear that most of the gains in speedup are from the re-estimation segment of the code. In particular, significant gains come from optimizing and parallelizing the code that is run with each IRLS iteration and Newton-Raphson phi re-estimation.

With the inference code, greater speedup will occur with larger state spaces. With GENSENG, the state space is limited to 7 possible states, which underutilizes the multiple cores. With significantly more states, then the multiple cores are not only all utilized but the cost of imperfect division of work is lessened. For example, if the calculations for each state require similar time to complete and GENSENG is run on 6 cores, then the calculations for the 7 states will be divided into a batch of 6 states and a second, underutilized batch of 1 state. Even still, parallelization results in 1.47 times total speedup for the code responsible for inference.

Within the re-estimation section, the calculations for the initial state probabilities require almost no time and therefore speedup is insignificant. The calculation of transition state probabilities sees a 3.906 times speedup, which is understandable considering that each state in each window is independent of every other state in every window and can be parallelized as such. When calculating the emission state probabilities, a speedup of 4.592 times can be seen. This speedup can largely be traced to `phi_ml`, which benefits from a special case of parallelization in its helper `score_info` code where reduction on a single register variable can be used.

The total speedup per round of training is about 4.462 times. Notably, the speedup in the total execution time is only 3.705 and this is due to the aforementioned issue of writing results to disk with every round of training. Eliminating the disk writes would presumably boost speedup to the higher per round level.

6. Other considerations

Parallelization depends not only on the number of processors but also efficient memory use. Within the parallelization code for GENSENG, many tradeoffs were made between prioritizing memory allocation, code readability and computation speed. This project attempted to make the optimal tradeoffs for most

machines, but fine-tuning could increase speedup for particular machines further. For example, the sequential implementation of GENSENG allocates memory to an array once and reuses the array multiple times for deallocation. In the parallel case, each thread may need the same array but each requires a copy. In these cases, the question is whether it is better to allocate the maximum amount of memory needed at once and require each thread to perform potentially complex index checks and calculations or to allocate memory on the fly for each thread. Clearly, such a decision depends on the specific hardware characteristics of the current machine. If the CPU of the current machine is fast relative to the memory, there is ample memory or the memory is slow to allocate, then it is reasonable to do index calculations. For this project, memory is allocated on the fly, which was shown to be faster with testing.

7. Conclusion

The goal of this project was to parallelize and speedup the performance of GENSENG. By identifying target code segments and systematically applying parallelization using OpenMP, a speedup of 3.7 times was achieved on an 8-core machine. The parallel implementation of GENSENG will run on both single and multi-core machines with no effect on accuracy or ease of use.

8. Sources

Abyzov A, Urban AE, Snyder M, Gerstein M. “CNVnator: an approach to discover, genotype, and characterize typical and atypical CNVs from family and population genome sequencing.” Genome Res. 2011 Jun; 21(6):974-84.

Aitman TJ. "Copy number polymorphism in Fcgr3 predisposes to glomerulonephritis in rats and humans". Nature. 2006. 439 (7078): 851–855.

Cappuzzo F, Hirsch, et al. (2005). "Epidermal growth factor receptor gene and protein and gefitinib sensitivity in non-small-cell lung cancer." Journal of the National Cancer Institute 97 (9): 643–655.

Cook EH, Scherer SW. "Copy-number variations associated with neuropsychiatric conditions". Nature, 2008; 455 (7215): 919–23.

Gonzalez E. “The Influence of CCL3L1 Gene-Containing Segmental Duplications on HIV-1/AIDS Susceptibility.” Science. 2005. 307 (5714): 1434–1440.

Iafrate AJ, Feuk L, Rivera MN, Listewnik ML, Donahoe PK, Qi Y, Scherer SW, Lee C. “Detection of large-scale variation in the human genome.” Nat. Genet. 2004; 36, 949–951. 19.

Laird, P.W. Principles and challenges of genomewide DNA methylation analysis. Nat. Rev. Genet. 2010; 11, 191–203. 17.

Rashid NU, Giresi PG, Ibrahim JG, Sun W and Lieb JD. “ZINBA integrates local covariates with DNA-seq data to identify broad and narrow regions of enrichment, even within amplified genomic regions.” Genome Biol. 2011; 12, R67.

Sebat J, Lakshmi B, Troge J, Alexander J, Young J, Lundin P, Maner S, Massa H, Walker M, Chi M. “Large-scale copy number polymorphism in the human genome.” Science. 2004; 305, 525–528.

St Clair D. “Copy number variation and schizophrenia.” Schizophr Bull. 2008. 35 (1): 9–12.

Stankiewicz P, Lupski JR. "Structural Variation in the Human Genome and its Role in Disease." Annual Review of Medicine. 2010; 61: 437–455.

Szatkiewicz JP, Wang W, Sullivan PF, Wang W, Sun W. “Improving detection of copy-number variation by simultaneous bias correction and read-depth segmentation.” Nucleic Acids Res. 2013 Feb 1;41(3):1519-32.