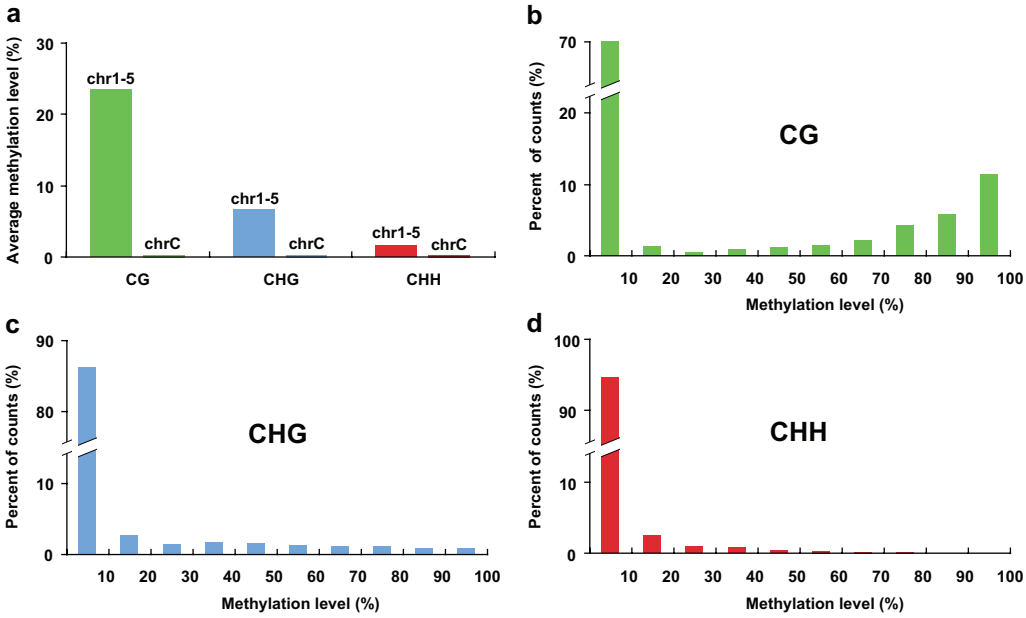
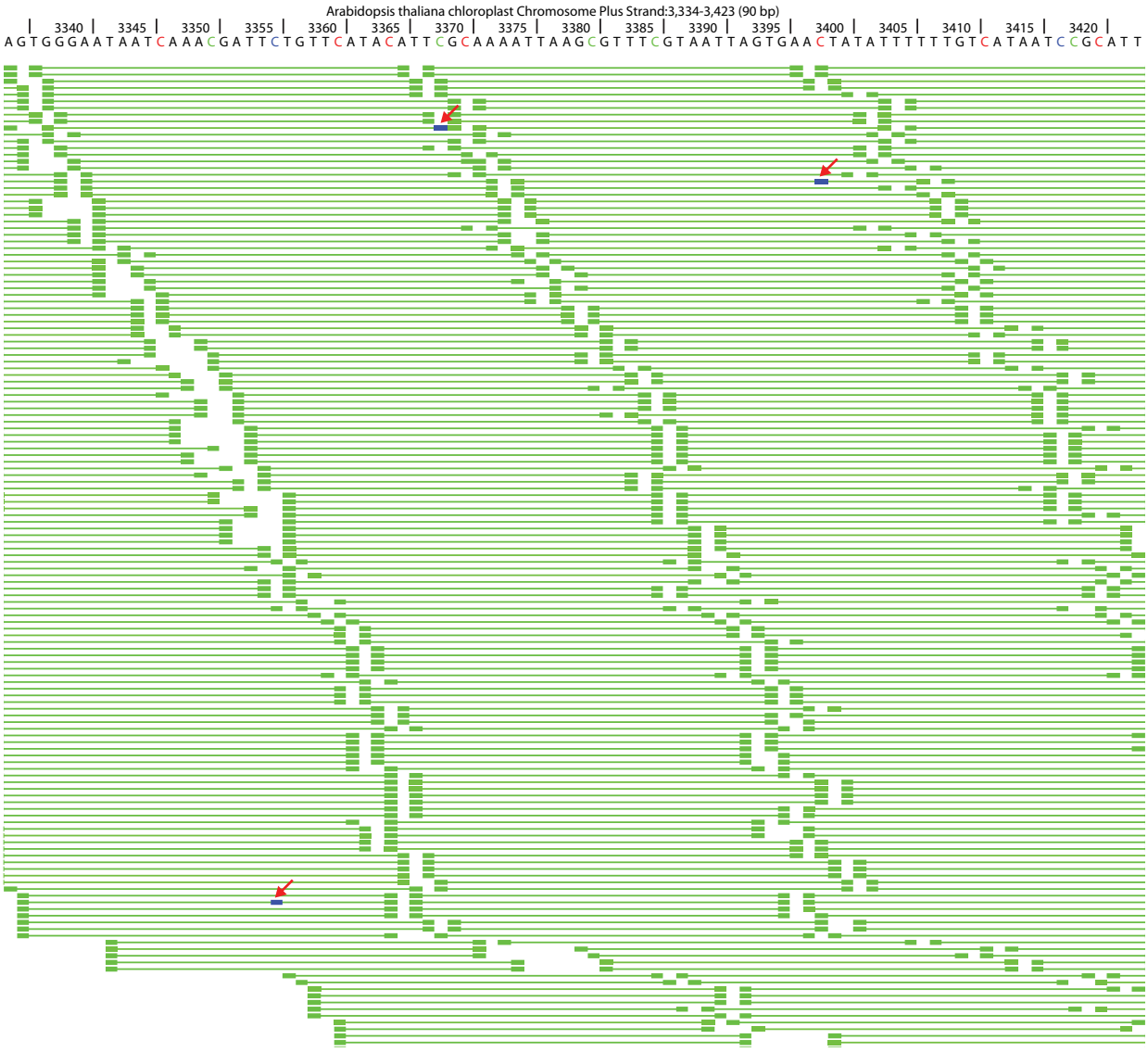
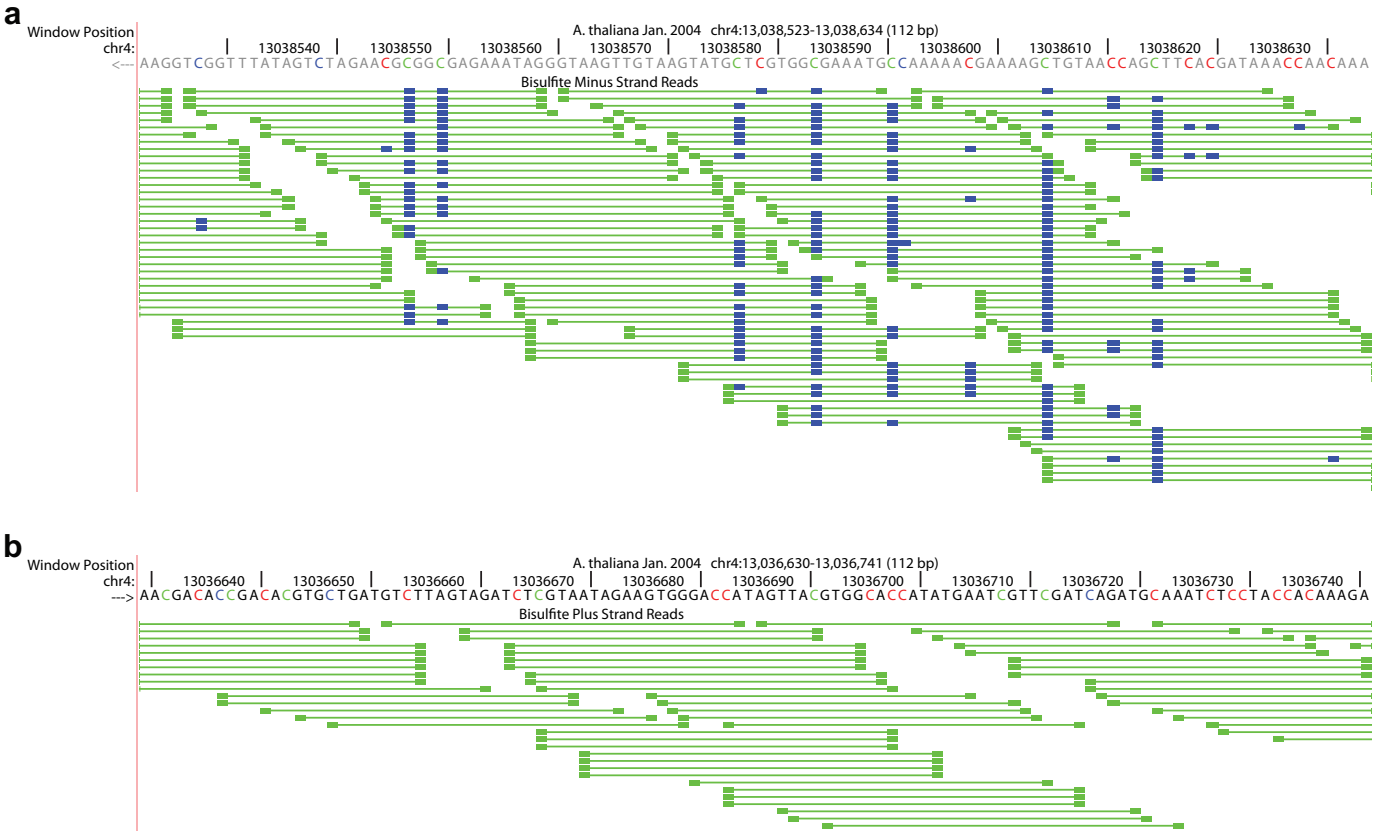
SUPPLEMENTARY INFORMATION



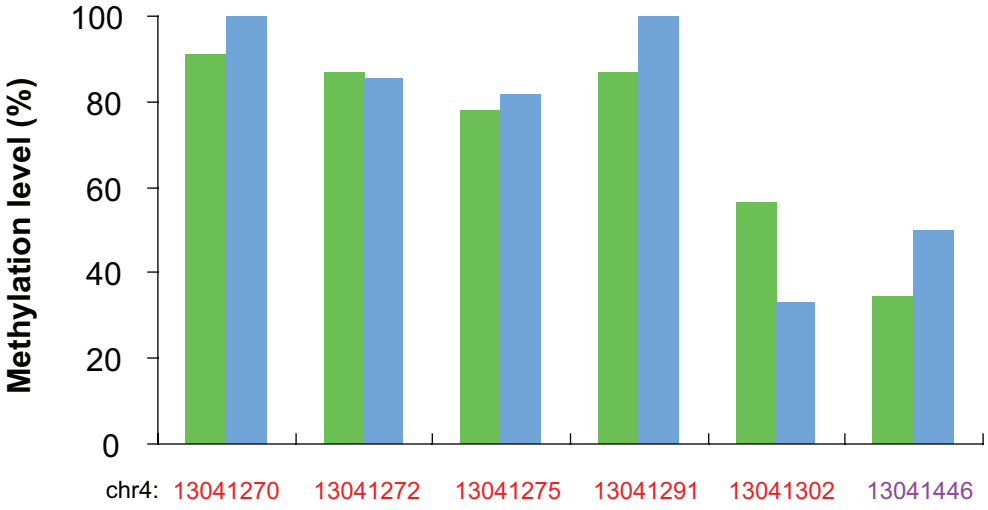
Supplementary Figure 1 | General patterns of methylation in the Arabidopsis genome. a, Genome average levels of methylation of the Arabidopsis nuclear genome (chromosome 1, 2, 3, 4, and 5) and the chloroplast genome (chromosome C). **b-d,** Distribution of methylation percentage of CG (**b**), CHG (**c**), and CHH (**d**). The *x*-axis is divided into 10 individual bins that correspond to methylation levels. The *y*-axis is the percent of total counts for each respective bin.



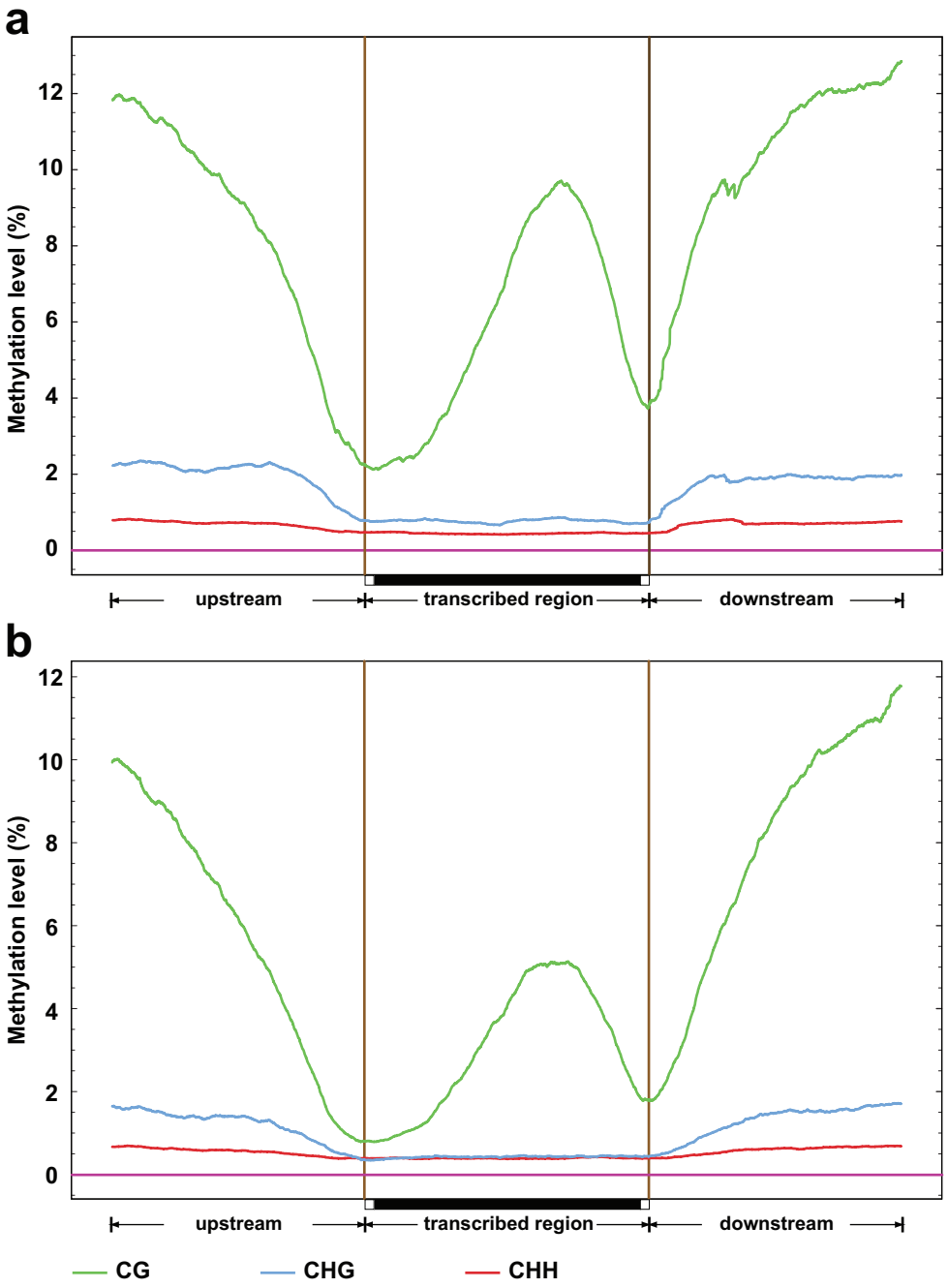
Supplementary Figure 2 | The chloroplast genome is unmethylated as shown by BS-Seq. BS-Seq data corresponding to nucleotides 3334 to 3423 of the chloroplast chromosome is displayed in detail. Cytosines in CG context are marked in green, in CHG context are marked in blue, and in CHH context are marked in red. BS-Seq reads that map to the plus strand are shown. Each read is represented by a thin green line that is aligned to its genomic location. Two green boxes mark the ends of each read and do not correspond to any sequence. A blue box in the middle of the reads (shown with red arrows) indicates a cytosine is detected in that position. Since the chloroplast genome is unmethylated, the occurrence of unconverted cytosines in the chloroplast genome reflects the background noise level of BS-Seq data.



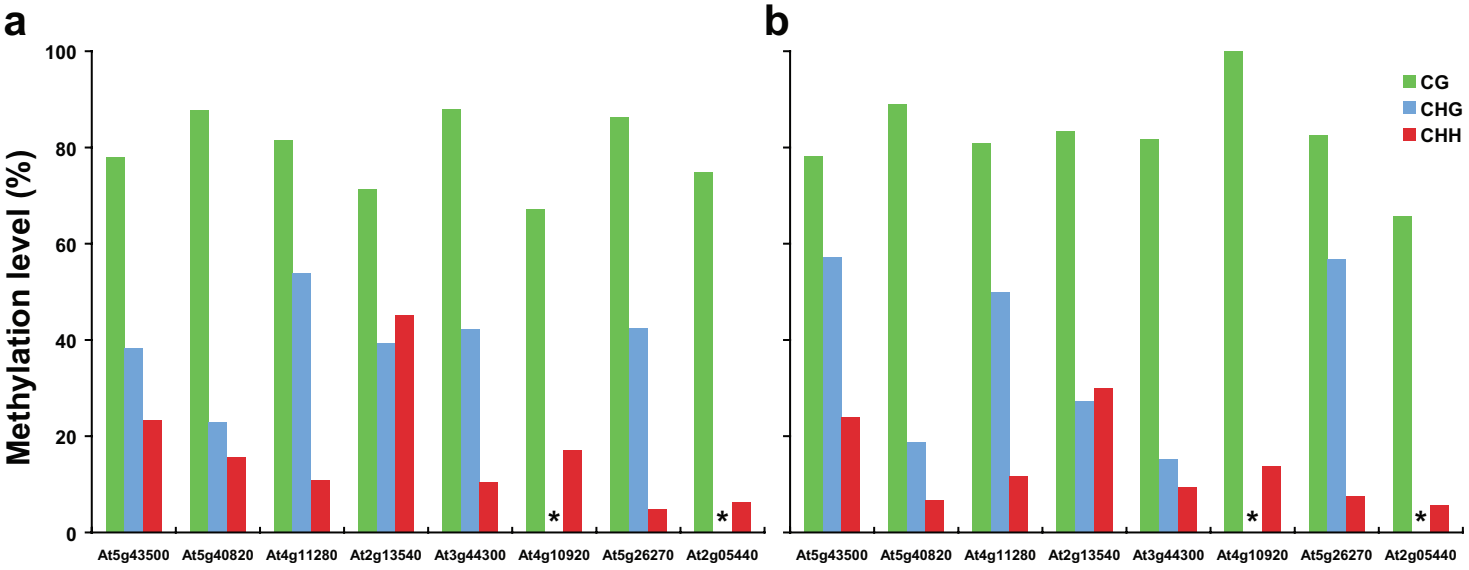
Supplementary Figure 3 | BS-Seq data from two selected regions within the *FWA* locus. BS-Seq reads mapped to a region (**a**) located within the known methylated *FWA* tandem repeats and to a known unmethylated region (**b**) upstream of the *FWA* tandem repeats (Zhang et al., Cell 126, 1189-1201) are shown. Genomic sequence of either plus strand (**b**) or minus strand (**a**) is displayed, with coordinates labeled on the top. Cytosines in CG context are marked in green, in CHG context are marked in blue, and in CHH context are marked in red. Each read is represented by a thin green line that is aligned to its genomic location. Two green boxes mark the ends of each read and do not correspond to any sequence. A blue box in the middle of a read indicates a cytosine is detected in that position.



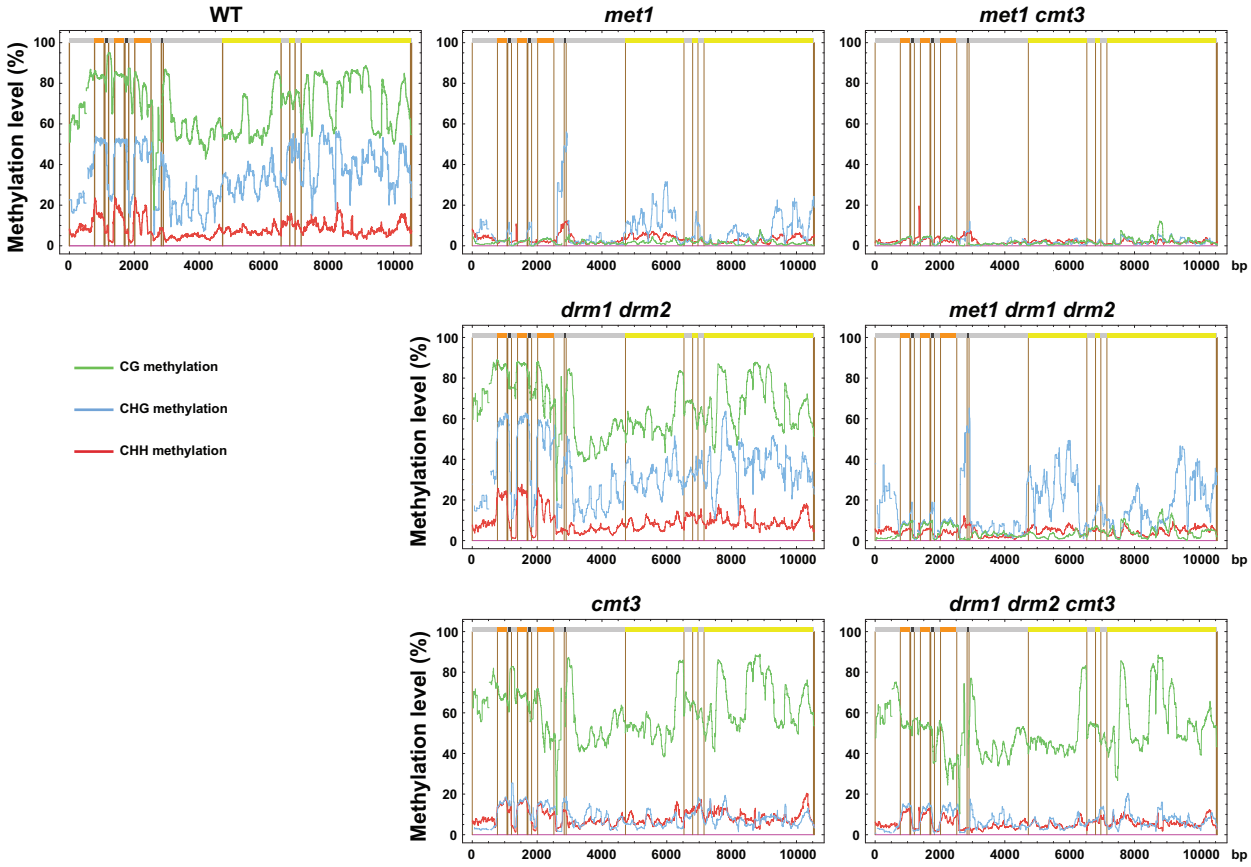
Supplementary Figure 4 | Comparison of BS-Seq data with traditional bisulfite sequencing data for selected CG sites at the *FWA* locus. We validated BS-Seq data (blue bars) by traditional bisulfite sequencing (green bars) at five consecutive CG sites (in red) and one lone CG site (in purple). Genomic coordinates of the cytosines in the CG sites are labeled on the bottom. PCR primers used are listed in Supplementary Table 7. Our previous microarray-based genomic methylation profiling study (Zhang et al., Cell 126, 1189-1201) did not detect methylation from any of the probes covering these CG sites. In another whole genome microarray-based genomic methylation profiling study (Zilberman et al., Nat Genet 39, 61-69), the single probe that overlaps directly with these CG sites has a $\log_2$(IP/input) value of –0.280, below the 0.34 cutoff value from that study for unmethylated probes. Similarly, in an McrBC-microarray study (Vaughn et al., PLOS Biol 5, e174), signal from a region containing these CG sites was also defined as unmethylated.



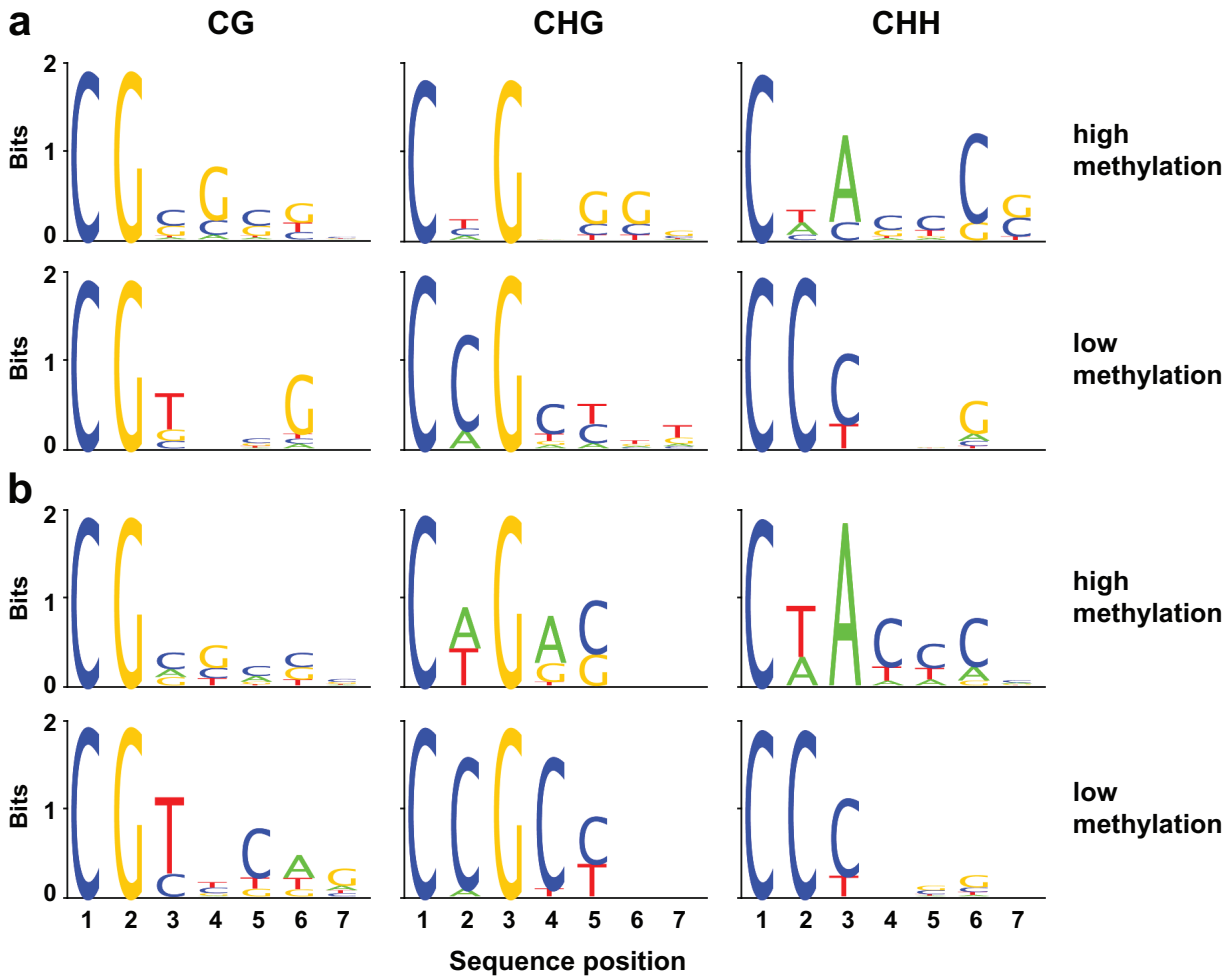
Supplementary Figure 5 | Detection of methylation by BS-Seq in genes previously classified as unmethylated. BS-Seq data mapping to genes previously classified as unmethylated by microarray studies (**a**: Zhang et al., Cell 126, 1189-1201; **b**: Zilberman et al., Nat Genet 39, 61-69) is plotted across gene bodies and upstream and downstream regions in 500 nucleotide sliding windows moving 5' to 3' (from left to right). The brown lines mark the boundaries between the upstream regions and gene bodies and between gene bodies and downstream regions. The purple line indicates zero percent methylation.



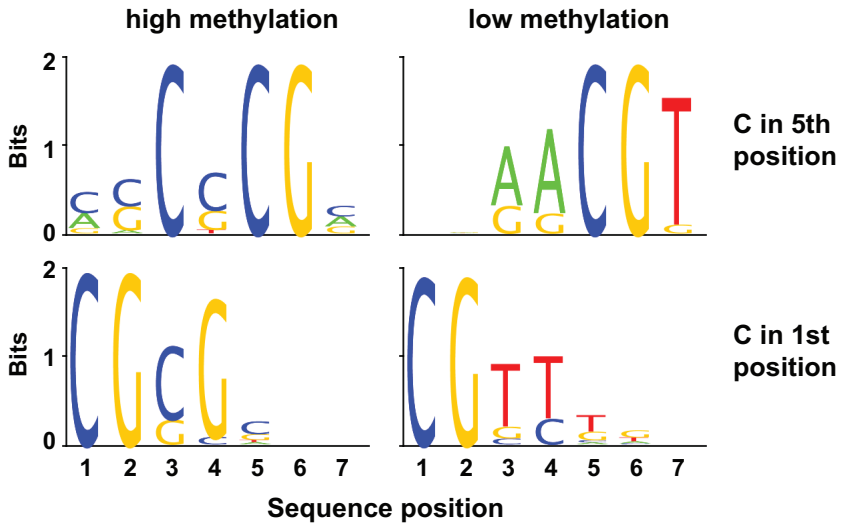
Supplementary Figure 6 | Validation of BS-Seq data in gene promoters by traditional bisulfite sequencing. Methylation status of a 235 bp region from the promoter of At5g43500, a 266 bp region from the promoter of At5g40820, a 295 bp region from the promoter of At4g11280, a 112 bp region from the promoter of At2g13540, a 320 bp region from the promoter of At3g44300, a 233 bp region from the promoter of At4g10920, a 258 bp region from the promoter of At5g26270, and a 153 bp region from the promoter of At2g05440 were examined by both traditional bisulfite sequencing (**a**) and BS-Seq (**b**). The asterisks indicate that there is no cytosine in the corresponding sequence context within the respective region. PCR primers used are listed in Supplementary Table 7. Among these genes, At4g11280, At3g44300, At4g10920, At5g26270, and At2g05440 are upregulated in *met1* and/or *drm1 drm2 cmt3* mutants, while others do not show altered expression in these methyltransferase mutants (Zhang et al., Cell 126, 1189-1201). At5g43500 and At2g13540 were previously classified as promoter methylated genes in our microarray-based methylation profiling study (Zhang et al., Cell 126, 1189-1201), whereas the other genes were not. We also checked the results from two other microarray-based methylation profiling studies (Zilberman et al., Nat Genet 39, 61-69; Vaughn et al., PLOS Biol 5, e174). In the Zilberman study, the regions of interest in At2g13540, At4g10920, and At2g05440 were not covered by any probes. One probe covered each of the regions of interest in At5g43500, At5g40820, At4g11280, At3g44300, and At5g26270, with $\log_2$(IP/input) values of 1.86, 1.89, 1.13, 1.37, and 1.62, respectively. The cutoff used for calling methylation in the Zilberman study was 1.28, so three of these regions would be classified as methylated, one would be close to the cutoff value, and one would be below the cutoff value. In the Vaughn study, only At4g10920 and At4g11280 were covered by the array. The 389 bp probe that covered the validated region of At4g10920 showed a robust methylation signal, while the 960 bp probe spanning At4g11280 showed a very weak methylation signal. Thus, the BS-Seq method detects methylation in regions that are otherwise variably detected in microarray studies and also provides quantitative data about methylation in different sequence contexts, e.g., CG, CHG, and CHH.



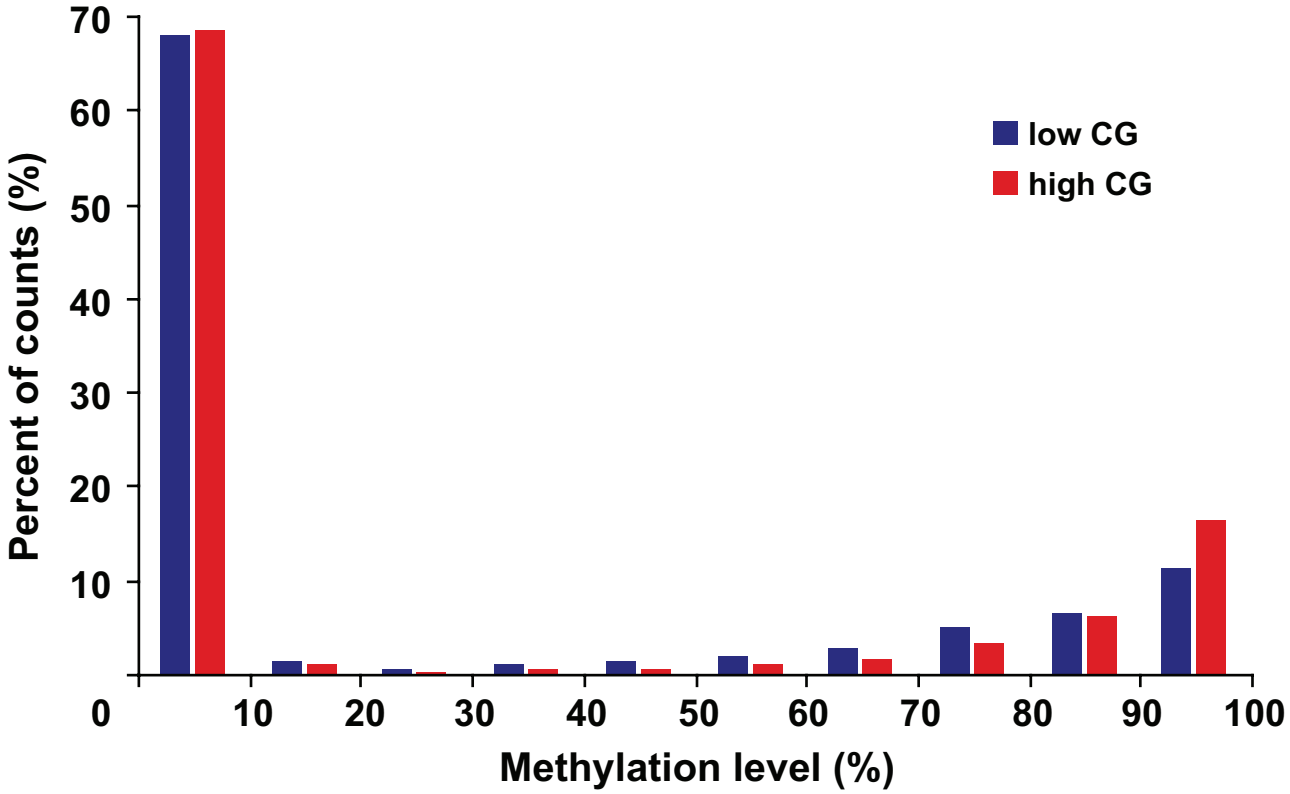
Supplementary Figure 7 | Detection of methylation in rDNA by BS-Seq.
Methylation level in rDNA is determined based on the BS-Seq reads that map to a canonical copy of the 45S rDNA. The purple line indicates zero percent methylation. Wild-type Arabidopsis and various methyltransferase mutants are analyzed as indicated. An illustration of 45S rDNA is shown at the top of each panel. The orange boxes indicate three SalI boxes. The black boxes indicate two spacer promoters (on the left) and the minimal promoter (on the right). Yellow boxes indicate 18S, 5.8S, and 25S loci (from left to right). Vertical brown lines mark the boundaries of the above-mentioned elements. The sequences and annotations of 45S rDNA were obtained from GenBank and several previous reports (GenBank Accession X15550 and AC006837; Gruendler et al., Nucleic Acids Res 17, 6395-6396; Doelling et al., Proc Natl Acad Sci USA 90, 7528-7532; Doelling and Pikaard, Plant J 8, 683-692).



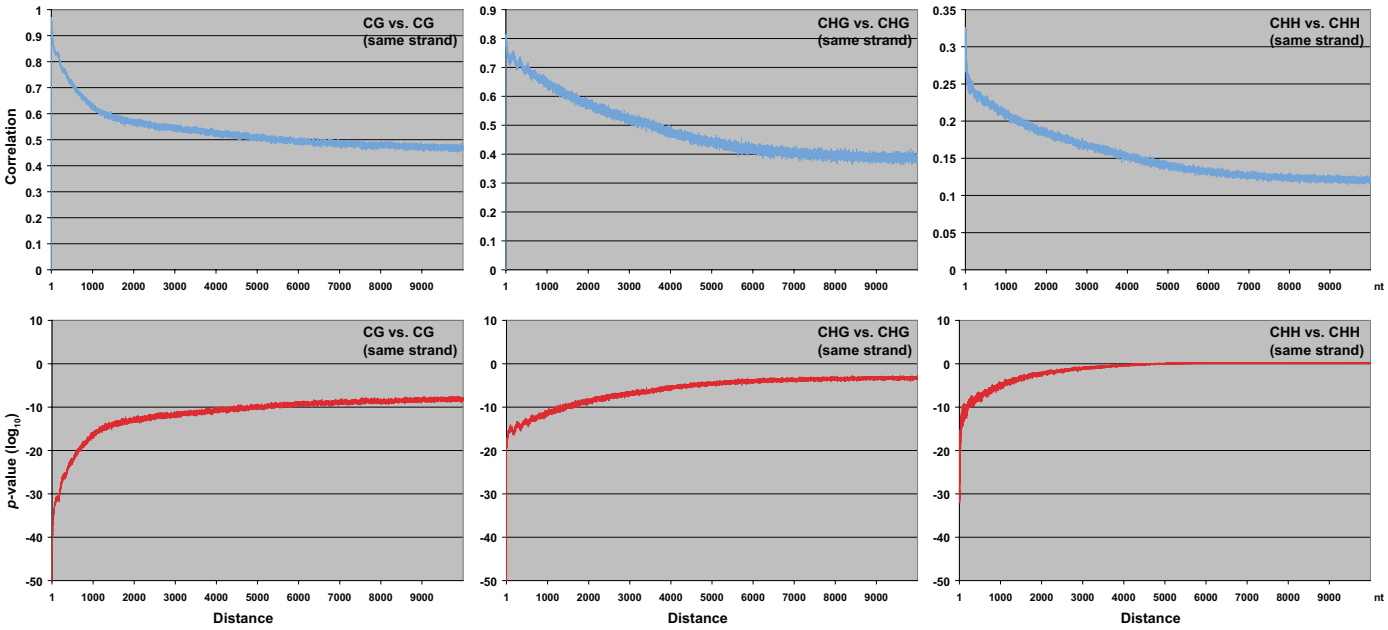
Supplementary Figure 8 | Sequence preferences for methylation in CG, CHG, and CHH contexts. Logos of sequence contexts that are preferentially methylated at the highest or lowest levels for 7-mer sequences in which the methylated cytosine is in the first position. In **a**, all genomic 7-mers in chromosome 1 were analyzed, while the sequences analyzed in **b** were restricted to previously-defined methylated sequences (Zhang et al., Cell 126, 1189-1201). The logo graphically displays the sequence enrichment at a particular position in the alignment of 7-mers in each class, measured in bits. The maximum sequence conservation per site is 2 bits (i.e., 1 base) when a site is perfectly conserved, and 0 if there is no preference for a nucleotide.



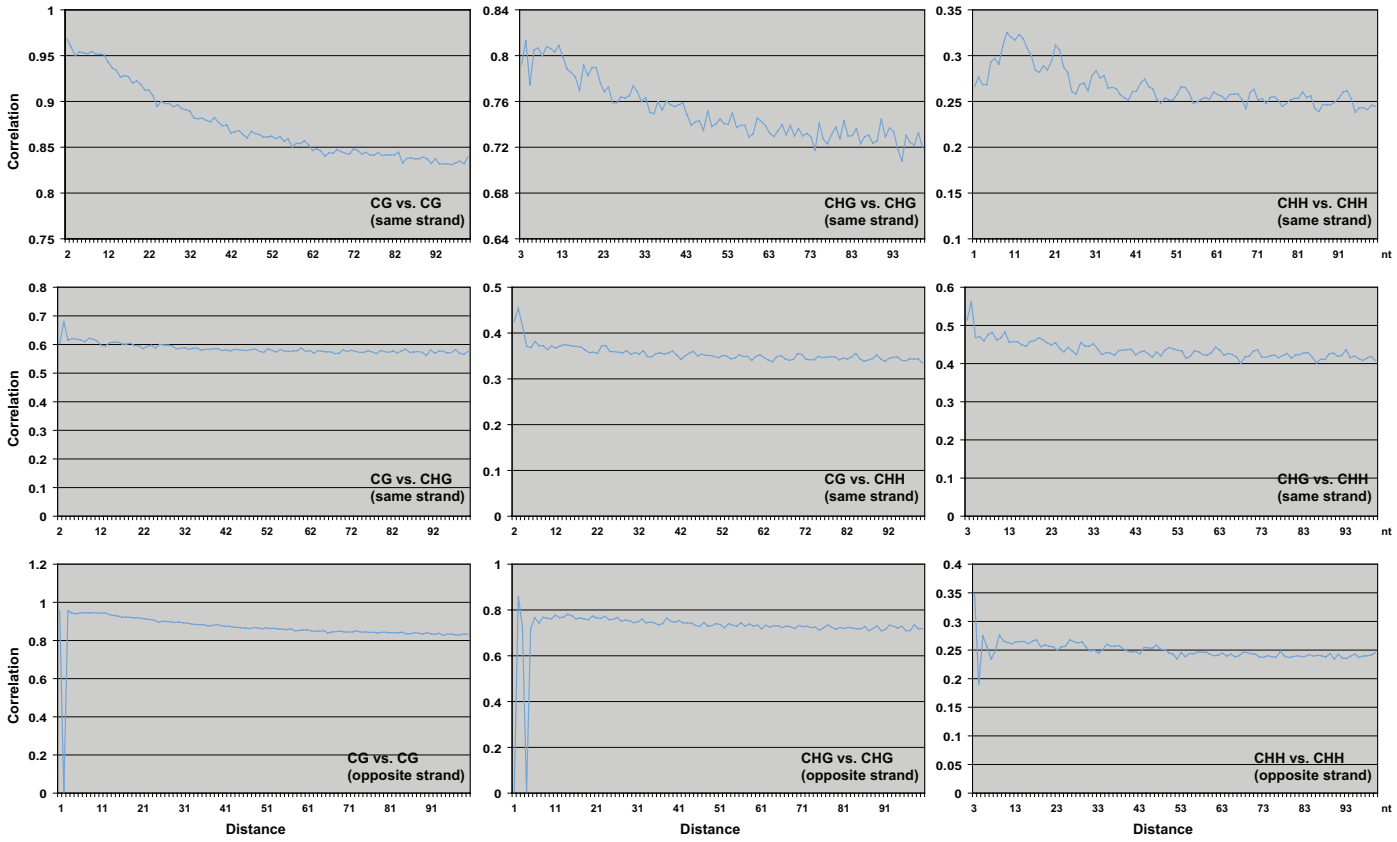
Supplementary Figure 9 | Sequence preferences for CG methylation within body methylated genes. Logos of sequence contexts that are preferentially methylated at the highest or lowest levels for 7-mer sequences in which the methylated cytosine is either in the fifth position or the first position. The sequences analyzed were restricted to previously-defined body methylated genes in the Arabidopsis genome (Zhang et al., Cell 126, 1189-1201). The logo graphically displays the sequence enrichment at a particular position in the alignment of 7-mers in each class, measured in bits. The maximum sequence conservation per site is 2 bits (i.e., 1 base) when a site is perfectly conserved, and 0 if there is no preference for a nucleotide.



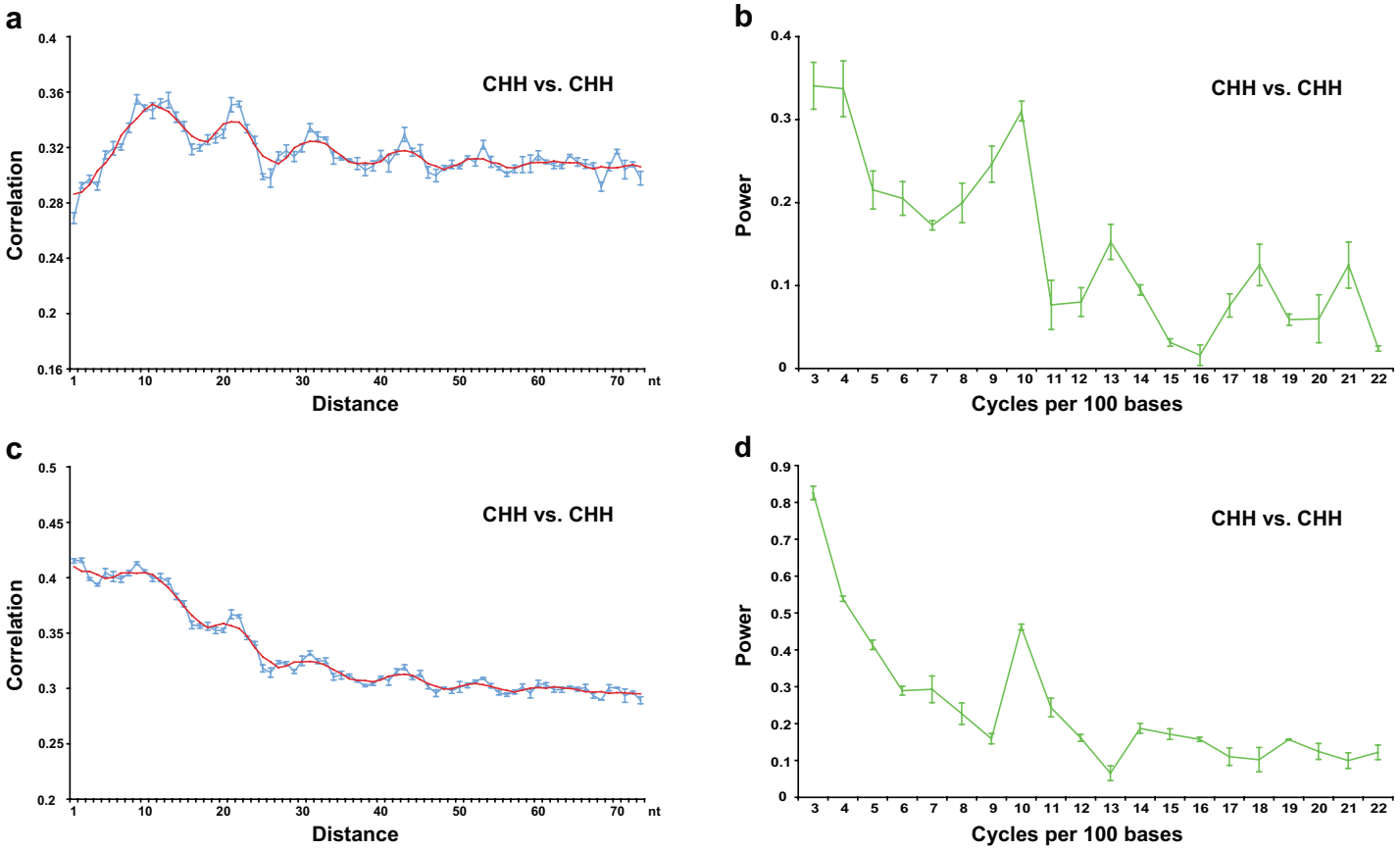
Supplementary Figure 10 | CG sites located in regions that contain a high local density of CG dinucleotides are more frequently methylated. Red bars (high CG) represent the methylation levels of CG sites at the center of 1,000 nucleotide windows that have more than 50 CG dinucleotides, and blue bars (low CG) represent CG sites at the center of 1,000 nucleotide windows that have fewer than 20 CG dinucleotides. The *x*-axis is divided into 10 bins that correspond to methylation levels. The *y*-axis is the percent of counts within each bin.



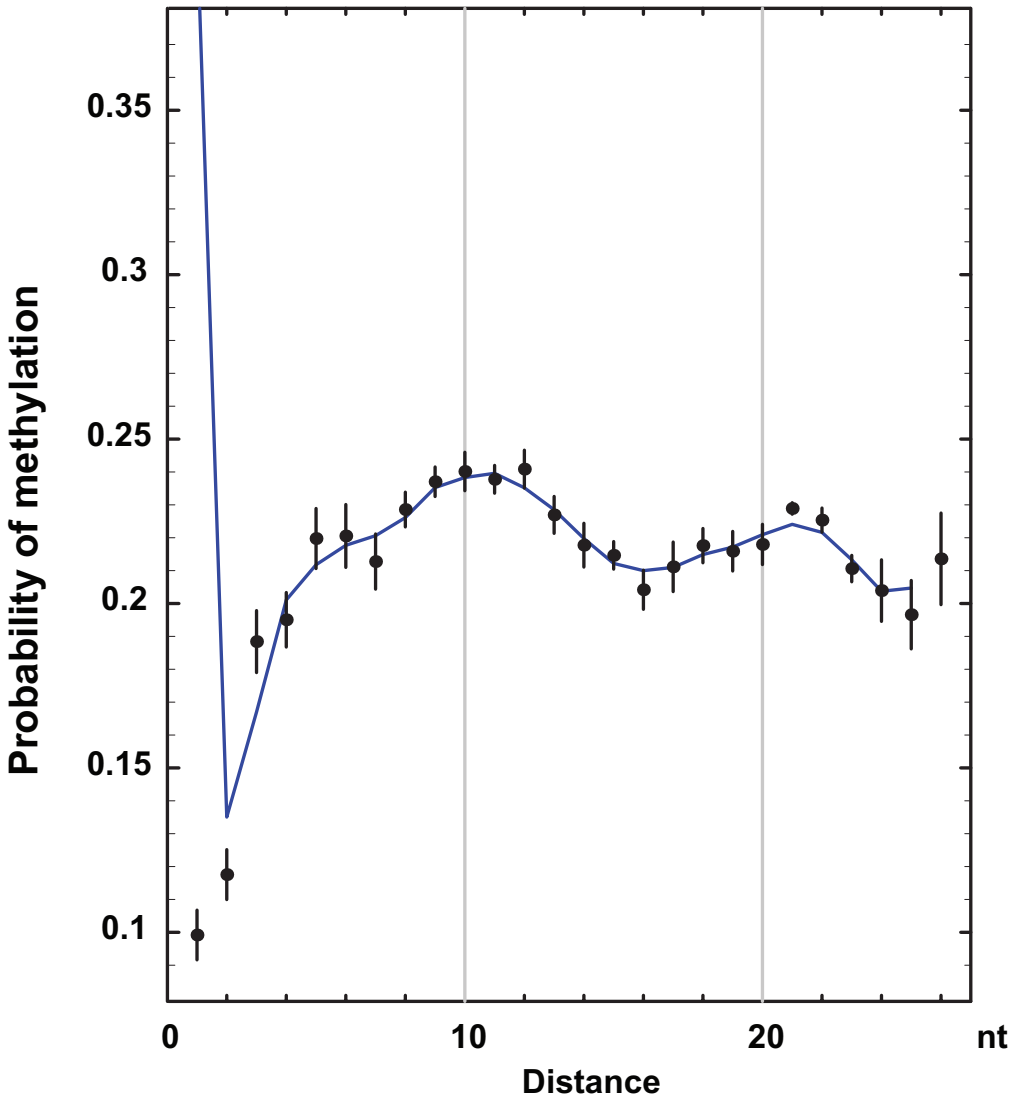
Supplementary Figure 11 | Autocorrelation of methylation between cytosines located within 10,000 nucleotides of each other. The *x*-axis indicates the distance between two cytosines. The *y*-axis in top panels indicates level of autocorrelation, and the *y*-axis in bottom panels indicates corresponding *p*-values (see Supplementary Table 5 for details).



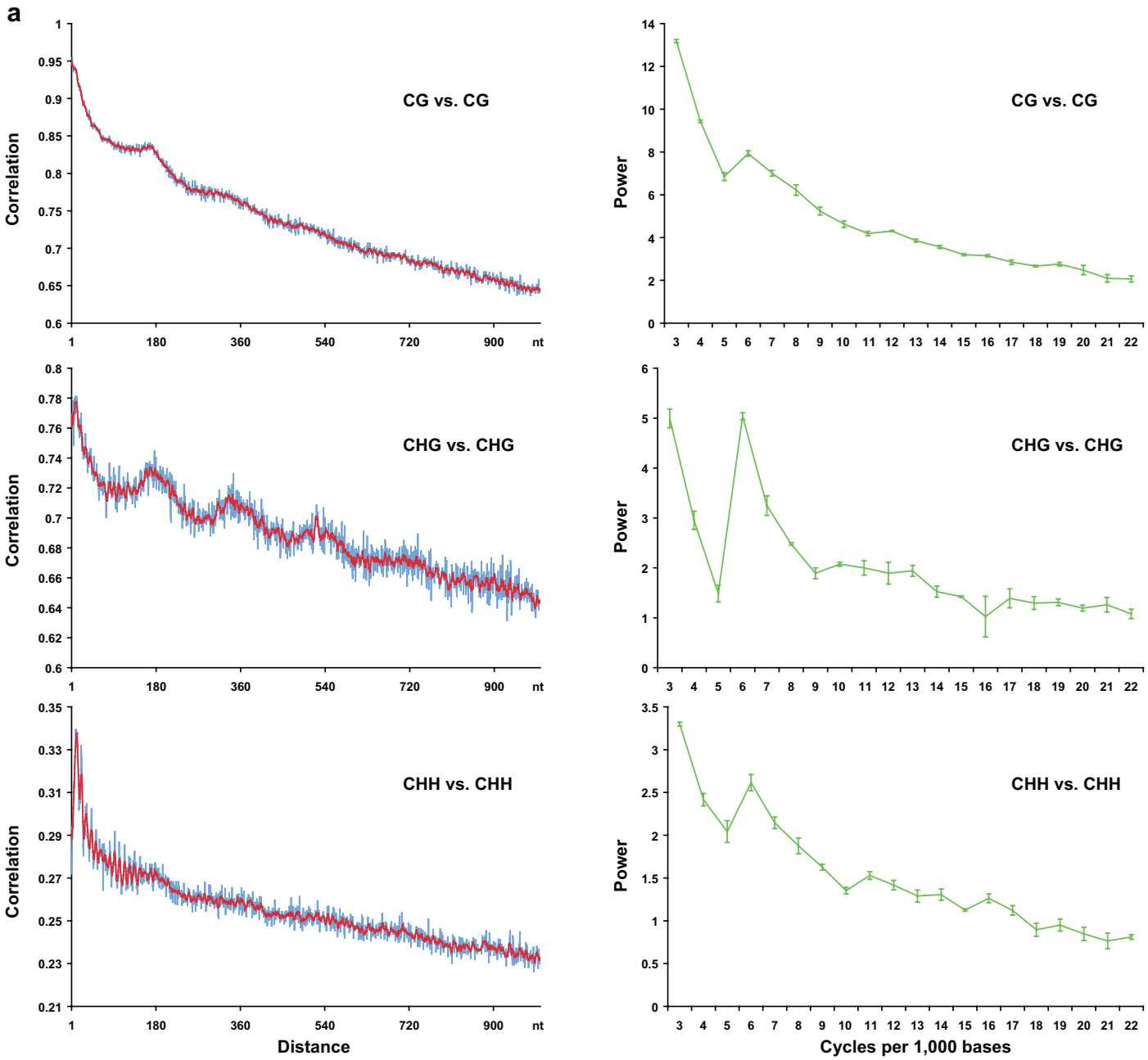
Supplementary Figure 12 | Autocorrelation of methylation between cytosines located within 100 nucleotides of each other. The *x*-axis indicates the distance between two cytosines. The *y*-axis indicates level of autocorrelation (see Supplementary Table 5 for details).



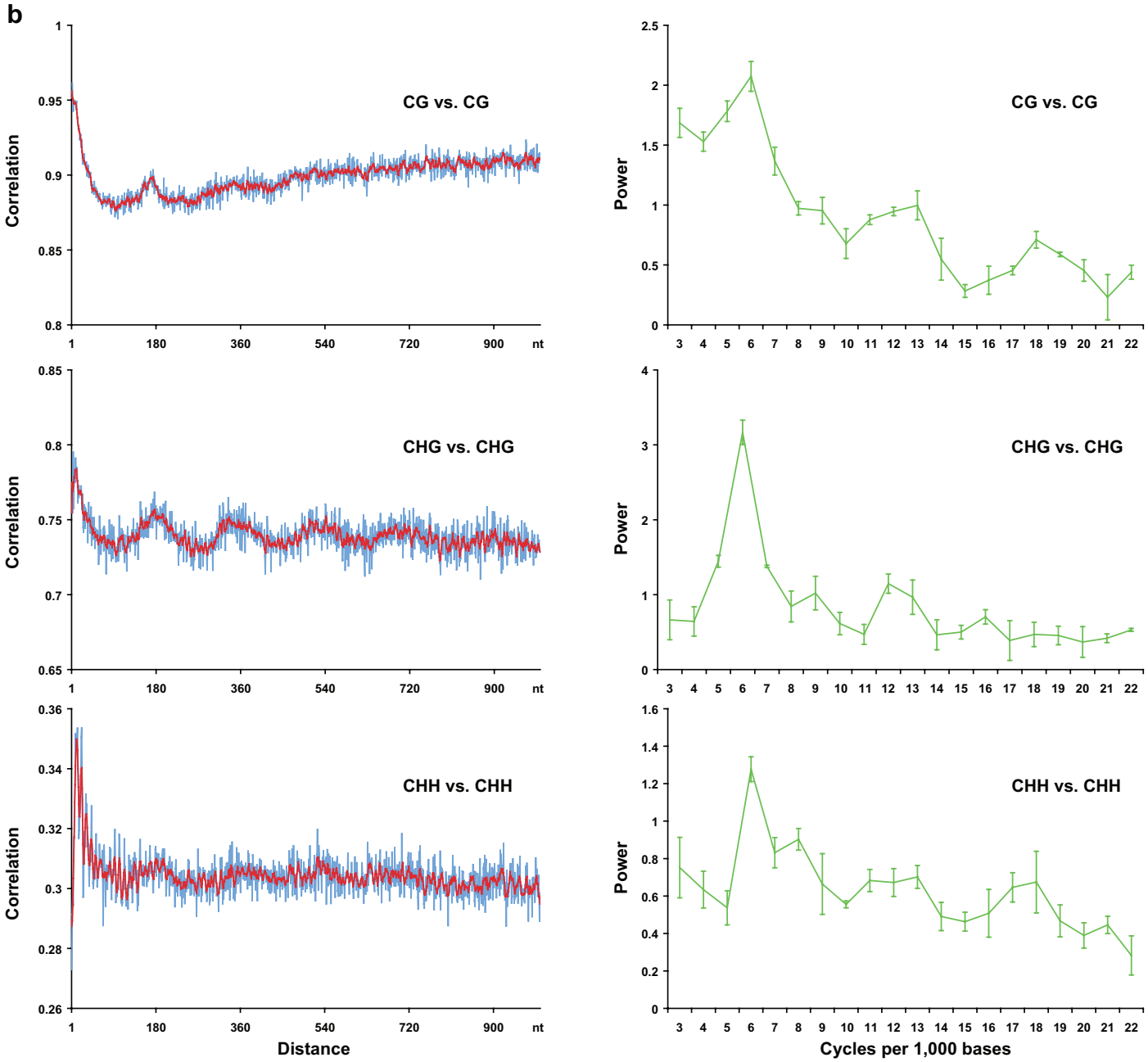
Supplementary Figure 13 | Detection of the periodic pattern of CHH methylation in various analyses. a, c, Autocorrelation of the methylation status of cytosines in a CHH context. The *x*-axis indicates the distance between the two cytosines. The *y*-axis indicates the level of autocorrelation in methylation. The red line is a running average of windows that are ±2 bases around a single base. **b, d,** Fourier transform analysis of CHH methylation correlation. The *x*-axis indicates the number of cycles per 100 bases. The *y*-axis is the amplitude of the corresponding frequency. The peak at position 10 represents a periodicity of ten nucleotides for high CHH methylation. The *p*-values for observing the peak Fourier transform values at position 10 by chance in random permutations of the genome sequence are below 10^{-106} (**b**) and 10^{-300} (**d**), respectively. In **a** and **b**, BS-Seq reads analyzed were restricted to those mapping to previously-defined methylated sequences (Zhang et al., Cell 126, 1189-1201). In **c** and **d**, BS-Seq reads, which otherwise would have been discarded by the non-conversion filter (see Methods: those with three consecutive methylated CHH sites), were included in the analyses. In **a**-**d**, Monte Carlo sampling of three datasets consisting of half the data was used to compute the mean and standard deviations of the autocorrelations and Fourier transforms. The mean values are shown. Error bars represent standard deviations. We also analyzed the autocorrelation of CHH sites thoughout the genome, regardless of methylation state, and found very low autocorrelation values and did not observe a 10 nucleotide periodicity.



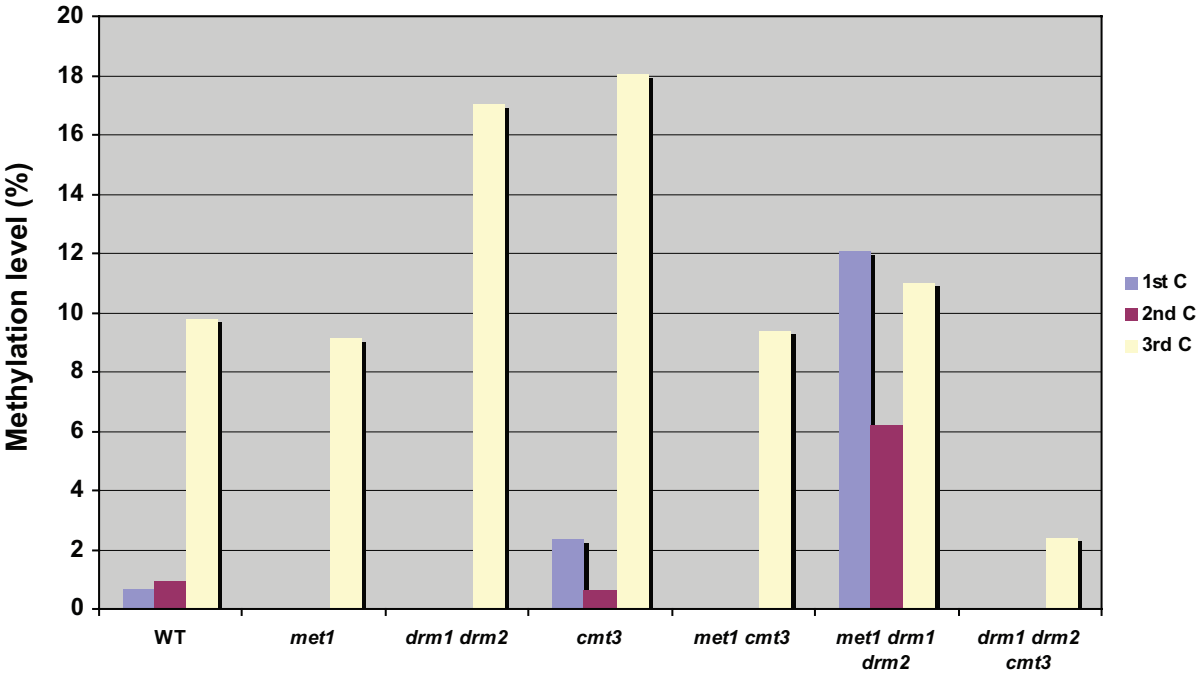
Supplementary Figure 14 | Within-read probability of additional methylation of CHH sites within a given distance from a methylated CHH site. Data were derived from individual BS-Seq reads. The *x*-axis indicates the distance between the two cytosines. The *y*-axis indicates the probability of methylation of CHH sites within the given distance from another methylated CHH site. Each point is the mean value from averaging the probability from each of the five Arabidopsis chromosomes, and the blue line is a running average of these mean values. Error bars represent 95% confidence intervals via critical values of Student *t* distributions.



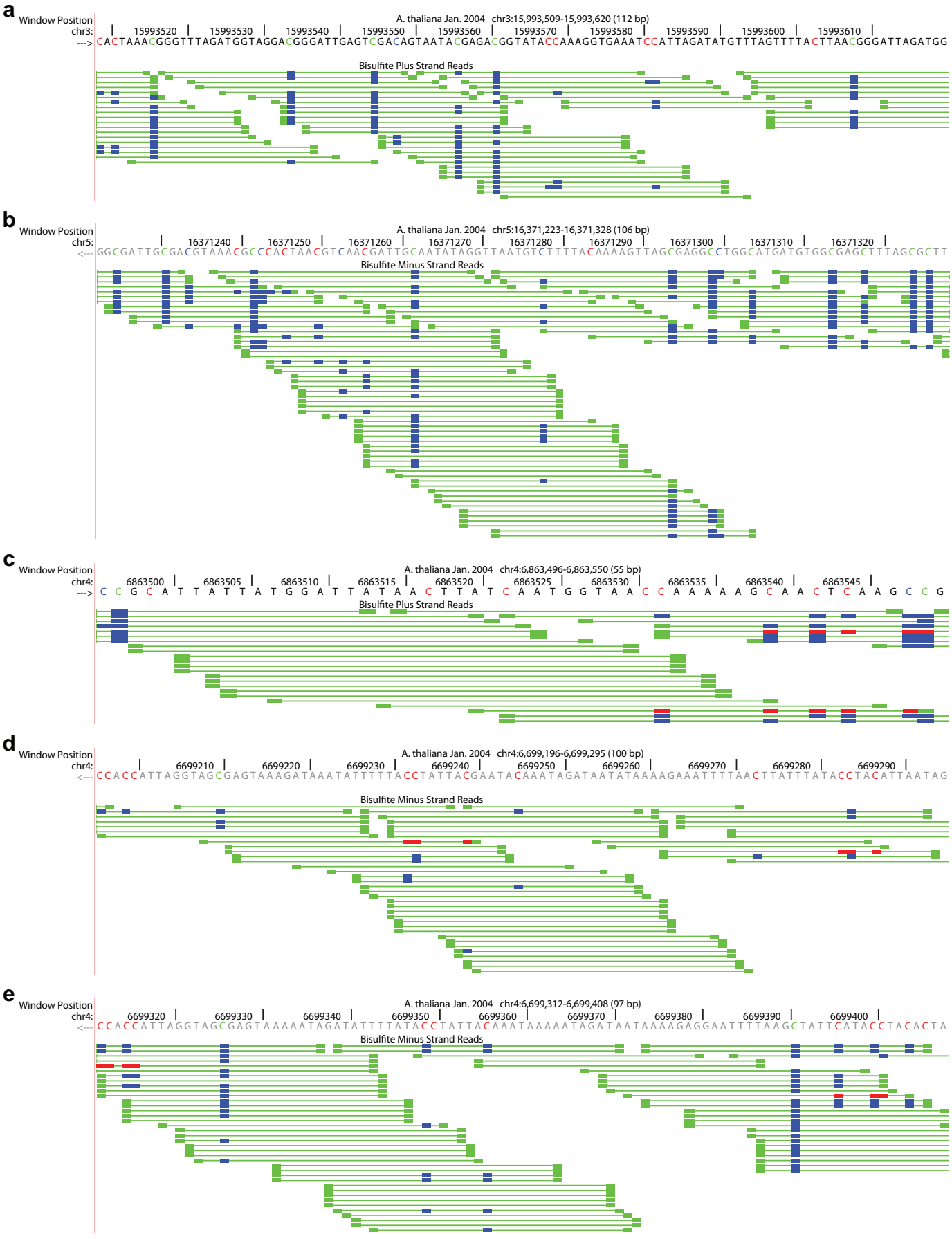
Supplementary Figure 15 (continued on next page)



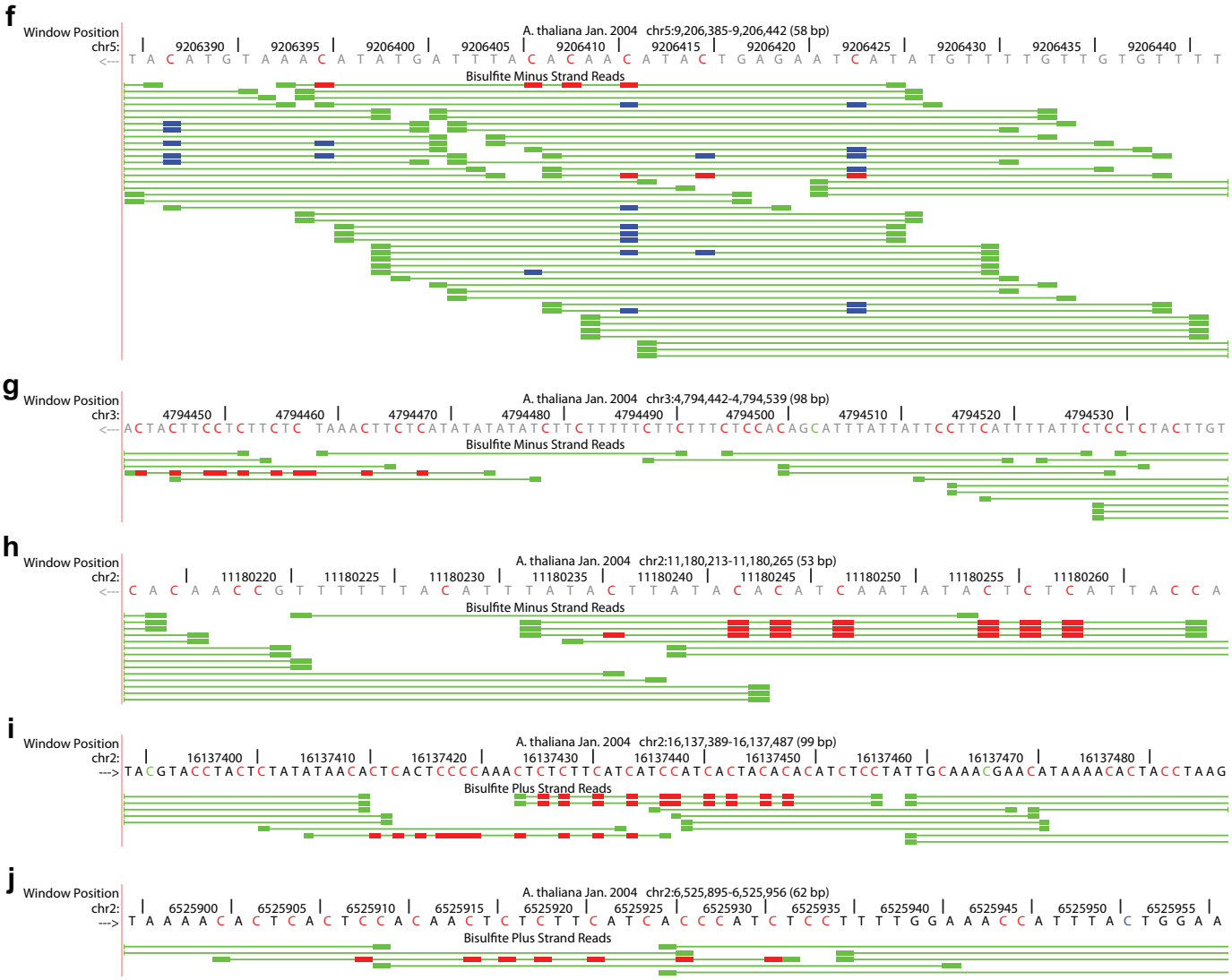
Supplementary Figure 15 | Periodic pattern of CG, CHG, and CHH methylation. Panels on the left show autocorrelation of the methylation status of cytosines in CG, CHG, and CHH context. The *x*-axis indicates the distance between the two cytosines. The *y*-axis indicates the level of autocorrelation in methylation. The red line is a running average of windows that are ±2 bases around a single base. Panels on the right are Fourier transform analyses of the correlation of CG, CHG, and CHH methylation. The *x*-axis indicates the number of cycles per 1,000 bases. The *y*-axis is the amplitude of the corresponding frequency. The peak at position 6 suggests a periodicity of approximately 167 nucleotides. The *p*-values for observing the peak Fourier transform values at position 6 by chance in random permutations of the genome sequence are below 10^{-66} (**a**, CG), 10^{-300} (**a**, CHG), 10^{-212} (**a**, CHH), 10^{-40} (**b**, CG), 10^{-251} (**b**, CHG), and 10^{-35} (**b**, CHH). In **a** and **b**, Monte Carlo sampling of three datasets consisting of half the data is used to compute the mean and standard deviations of the autocorrelations and Fourier transforms. The mean values are shown. Error bars in Fourier transform analysis graphs represent standard deviations. In **a**, methylation from the whole genome was analyzed, while in **b** the analysis was restricted to previously-defined methylated sequences (Zhang et al., Cell 126, 1189-1201).



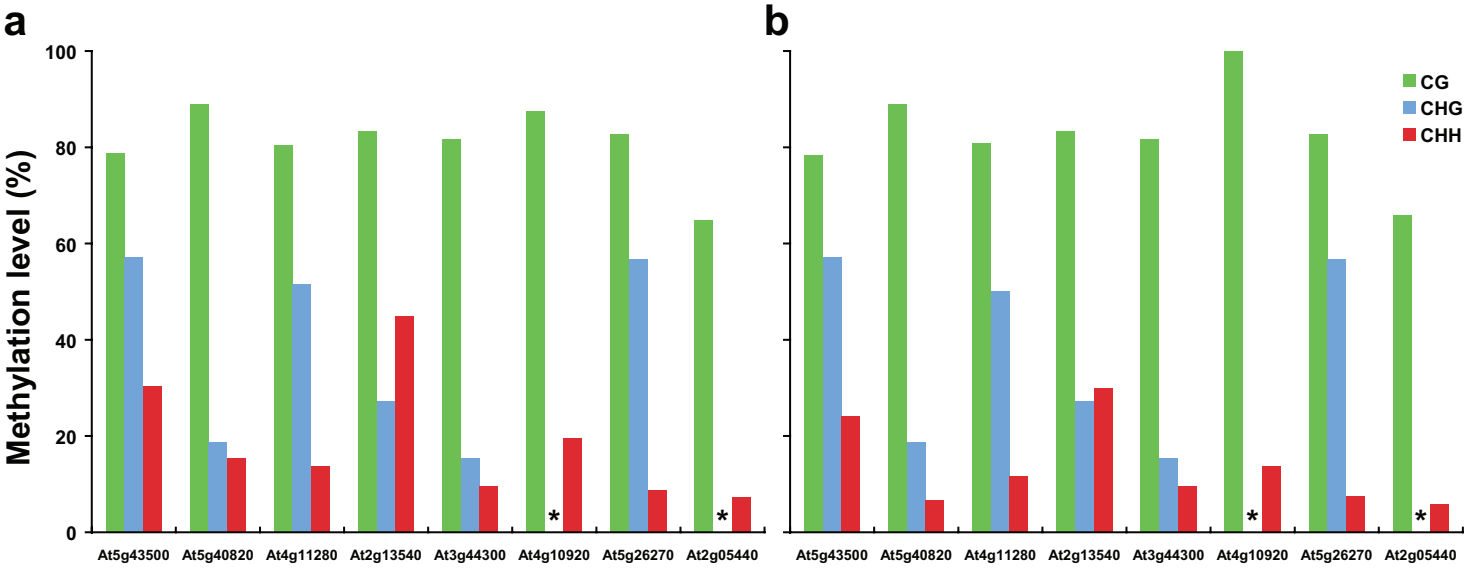
Supplementary Figure 16 | Detection of telomere methylation by BS-Seq in wild type and methyltransferase mutants. Methylation levels of the three consecutive cytosines in the $(CCCTAAA)_n$ telomeric repeat are calculated in wild type and various Arabidopsis methyltransferase mutants.



Supplementary Figure 17 (continued on next page)



Supplementary Figure 17 | Examples of applying non-conversion filtration to BS-Seq reads. BS-Seq data of ten selected regions is shown. Genomic sequence of either plus strand or minus strand is displayed, with coordinates labeled on the top. Cytosines in CG context are marked in green, in CHG context are marked in blue, and in CHH context are marked in red. Each read is represented by a thin green line that is aligned to its genomic location. Two green boxes mark the ends of each read and do not correspond to any sequence. A blue or red box in the middle of a read indicates a cytosine is detected in that position. Reads with red boxes are discarded by the non-conversion filter, which eliminates any read containing three or more consecutive methylated CHH sites. Regions in **a**-**f** were part of the traditional bisulfite sequencing validation regions of Supplementary Fig. 6, and regions in **g**-**j** were validated by traditional bisulfite sequencing to be unmethylated. (PCR primers used are listed in Supplementary Table 7.) These examples demonstrate that the non-conversion filter effectively reduces false positive methylation signals generated from DNA fragments that are not properly bisulfite converted (**g**-**j**), and also reduces to some extent the estimate of the overall level of CHH methylation in regions containing heavy CHH methylation (**c**-**f**).



Supplementary Figure 18 | Analyzing the effect of non-conversion filtration at selected loci. BS-Seq data before (**a**) and after (**b**) applying the non-conversion filter are shown for the regions analyzed in Supplementary Fig. 6. The asterisks indicate that there is no cytosine in the corresponding sequence context within the respective region.

SUPPLEMENTARY METHODS

DNA Sample Preparation

Wild type Arabidopsis plants used in this study are of the Columbia-0 ecotype. Methyltransferase mutants *met1*, *cmt3*, *drm1 drm2*, *met1 cmt3*, *met1 drm1 drm2*, and *drm1 drm2 cmt3* were previously described [Zhang, X. and Jacobsen, S. E. Cold Spring Harb. Symp. Quant. Biol. 71, 439–47 (2006)]. Five–week–old continuous light–grown plants were utilized for this study.

For nuclear DNA isolation, 1 g plant tissues were ground into a fine powder in liquid nitrogen, homogenized in 10 ml HBM buffer (25 mM Tris-Cl [pH 7.6], 0.44 M sucrose, 10 mM $MgCl_2$, 0.1% Triton X-100, 10 mM beta-mercaptoethanol, 2 mM spermine, 1 mM PMSF, 1 μg/ml pepstatin, 1X EDTA–free protease inhibitors [Roche]). After filtering through Miracloth (Calbiochem) twice, the homogenate was centrifuged, and the pellet was resuspended in 5 ml HBB buffer (25 mM Tris-Cl [pH 7.6], 0.44 M sucrose, 10 mM $MgCl_2$, 0.1% Triton X-100, 10 mM beta-mercaptoethanol). Subsequently, the sample was loaded onto 30 ml 40%/60% percoll gradient (made by mixing percoll and HBB buffer) and centrifuged to pellet nuclei. The nuclei were then washed twice by 10 ml HBB buffer, resuspended in 500 μl 50 mM Tris-Cl (pH 7.5) containing 20 μl Proteinase K (Roche) and incubated at room temperature for 30 minutes. Next, 1 ml 50 mM Tris-Cl (pH 7.5) containing 15 mM EDTA and 1.5% SDS was added to lyse the nuclei. Finally, DNA was purified by phenol/chloroform extraction and ethanol precipitation.

To obtain fragmented DNA of the desired size range, 5 μg of DNA was sonicated by Biorupter (Diagenode) for 60 minutes (four times at 15 minutes each). The ends of the DNA fragments were modified by sequential treatment with T4 DNA polymerase, Klenow DNA polymerases, T4 polynucleotide kinase, and Klenow DNA polymerase (3′→5′ exo$^-$) to generate blunt–ended DNA with 5′ phosphorylation and single "A" base 3′ overhang. The mouse libraries described in Figure 4c and Supplementary Table 1 were made from 5 μg of DNA extracted from either embryonic stem cells of different genotypes or male premeiotic germ cells of 13–14 day old pups.

Bisulfite Treatment

Double–stranded DNA adaptors containing the DpnI restriction site were ligated to the end–modified DNA fragments. After purifying away free adaptors by gel electrophoresis, DNA fragments were subjected to bisulfite treatment by a CpGenome DNA modification kit (Chemicon) in the presence of urea. The resulting DNA was used as template in PCR amplification to obtain double–stranded DNA, following conditions from a previous study [Meissner, A. et al. Nucleic Acids Res. 33, 5868–77 (2005)], except that extension temperature was set to 60°C instead of 72°C. To ensure selectivity, PCR primers were designed to only amplify DNA with bisulfite–converted adaptor sequences at both ends. Following PCR, adaptors were digested away by DpnI restriction enzyme leaving five basepairs of adaptor sequence on each end.

Library Generation and Illumina/Solexa Sequencing

DpnI digested DNA was incubated with Klenow polymerase (3′→5′ exo$^-$) to extend a single "A" base at 3′ ends. Double–stranded DNA adaptors (Illumina) were ligated to the DNA

fragments. Ligation products were run on 2% agarose gel and DNA of the size between 120 and 170 bp was recovered from the gel. The adaptor–ligated DNA was amplified using PCR primers 1.1 and 2.1 (Illumina). The PCR reaction was carried out per manufacturer instructions (Illumina), except that Pfu Turbo Cx Hotstart DNA polymerase (Stratagene) was used and annealing and extension temperatures were both set to 60°C. Purified PCR products were directly sequenced using an Illumina 1G Genome Analyzer following manufacturer protocols, generally yielding 36 nucleotides of sequence per cluster, generally resulting in 31 nucleotides of genomic sequence per cluster after the analysis pipeline trims the five initial non–genomic bases resulting from the library construction procedure.

Detection of Methylation Levels at Individual Loci

Bisulfite sequencing of selected regions was performed as previously described [Cao, X. and Jacobsen, S. E. Proc. Natl. Acad. Sci. USA 99 Suppl. 4, 16491–8 (2002)], except that CpGenome DNA modification kit (Chemicon) was used. The primers used for amplification of converted DNA after bisulfite treatment are listed in Supplementary Table 7.

Mathematical Abstraction of Sequencing Data

The abstraction used here for sequencing data is as follows. Sequencing experiments produce non-empty finite unordered lists (i.e., non-empty finite multisets) of *reads*. Each read of integer *length* $n \geq 0$ nucleotides (for Solexa, n is typically a few dozen) is given as a 4–by–n matrix $p_{i,j}$ of non-negative (and, in the present application, all positive) real numbers with every column sum equal to 1. For every j in $1..n$, the column $(p_{1,j}, p_{2,j}, p_{3,j}, p_{4,j})$ is interpreted as an independent distribution (probability "A", probability "C", probability "G", probability "T") for the jth base starting at the 5′ end of the putative DNA fragment of interest corresponding to the read. In other words, each read is viewed as a *position–weight matrix*. Ideally, each column would contain a single 1 and three zeros, but there is uncertainty due to sequencing being a physical measurement process.

Reads of length n can also be viewed naturally as certain probability distributions on the set Σ^n of strings of length n over the alphabet {A, C, G, T}. If $s := s_1 s_2 \ldots s_n$ is such a string (e.g., 5′–GATTACA–3′) with the encoding A=1, C=2, G=3, and T=4 (e.g., $s_1 = 3$, …, $s_7 = 1$), then the probability of s is

$$\Pr(s \mid \text{read}) = \prod_{j=1}^{n} p_{s_j, j} \in [0, 1].$$

The entropy of this probability distribution provides a natural measure of read "fuzziness" (basecall imprecision). This entropy, in bases, is

$$H(\text{read}) = \sum_{s \in \Sigma^n} f\left(\Pr(s \mid \text{read})\right) \in [0, n]$$

where $f(x) := -x \log_4 x \geq 0$ for x in (0, 1] is extended by continuity to x in [0, 1]. The number of *effective sequenced bases* is then

$$\mathrm{ESB}(\mathrm{read}) := n - H(\mathrm{read}) = n - \sum_{j=1}^{n} \sum_{i=1}^{4} f(p_{i,j}),$$

which varies between zero (when every base is completely unknown: probability "A" = probability "C" = probability "G" = probability "T" = 1/4) and n (when every base is known with certainty: every column of the matrix for the read has a single 1 and three zeros).

Depending on whether a read sequences a molecule whose history involves an even or odd number of DNA replication–based amplification steps from some reference DNA fragment, it may be necessary to *reverse complement* a read, an operation which preserves read length. In such cases, the matrix for the reverse complement is given by $q_{i,j} := p_{5-i,\ n-j+1}$ for i in 1..4 and j in 1..n.

Mathematical Abstraction of Genomes

Sets of genomic windows are abstracted in a similar manner as reads. Here, a *genome* is considered to be a non-empty finite list of *genomic sequences*. These are non-empty finite strings over the IUPAC nucleotide alphabet {A, B, C, D, G, H, K, M, N, R, S, T, V, W, Y} that enumerate the + and – strand 5′ to 3′ sequences of the chromosomes (and mitochondrion, chloroplast, …) that comprise the genome, generally as best as they are currently known.

Each IUPAC nucleotide letter is taken to represent a probability distribution on "A", "C", "G", and "T" as to the true nucleotide for that genomic position. From the original papers describing the sequences for the *Arabidopsis* chromosomes [Theologis et al., Nature, 2000 Dec. 14;408(6814):816–20; Lin et al., Nature, 1999 Dec. 16;402(6763):731–2; Salanoubat et al., Nature, 2000 Dec. 14;408(6814):820–2; Mayer et al., Nature, 1999 Dec. 16;402(6763):769–77; Tabata et al., Nature, 2000 Dec. 14;408(6814):823–6], estimates for accuracy of sequence deemed "finished" varies between 1 error in 10,000 basepairs and 1 in 300,000. For both *Arabidopsis* ATH1 chr1–5/C/M and NCBI 37.1 assembled mouse chr1–19/X/Y genomes, a uniform 0.9999 probability distributed equally among the members of the subset of {A, C, G, T} consistent with the stated discrete IUPAC basecall was adopted, with the remaining 0.0001 distributed equally among the members of the remaining subset of {A, C, G, T} (except for "N", for which (1/4, 1/4, 1/4, 1/4) was the adopted probability distribution).

Given $n \geq 1$, an *n–mer window* into a given genome is any sequence of n contiguous positions ordered 5′ to 3′ from any single one of the genome's genomic sequences. (Generally, n is much, much shorter than the lengths of whole genomic sequences — typically there are hundreds of millions or billions of windows; it is assumed there is at least one.) The sequence of each window is considered to be in the same format as a read, that is, as a position–weight matrix, a probability distribution on Σ^n, etc.

Comparison of Matrices

Given two position–weight matrices p and q of the same length n (typically with p being from a genomic window and q being a read), the probability that the sequence s from the first is the same as the sequence t from the second is

$$S^*(p,q) = \prod_{j=1}^{n} \Pr\left(s_j = t_j\right) = \prod_{j=1}^{n} \begin{pmatrix} \Pr(s_j = t_j = \text{“A”}) + \Pr(s_j = t_j = \text{“C”}) + \\ \Pr(s_j = t_j = \text{“G”}) + \Pr(s_j = t_j = \text{“T”}) \end{pmatrix}$$
$$= \prod_{j=1}^{n} \begin{pmatrix} \Pr(s_j = \text{“A”})\Pr(t_j = \text{“A”}) + \Pr(s_j = \text{“C”})\Pr(t_j = \text{“C”}) + \\ \Pr(s_j = \text{“G”})\Pr(t_j = \text{“G”}) + \Pr(s_j = \text{“T”})\Pr(t_j = \text{“T”}) \end{pmatrix}$$
$$= \prod_{j=1}^{n} (p_{1,j}q_{1,j} + p_{2,j}q_{2,j} + p_{3,j}q_{3,j} + p_{4,j}q_{4,j}) \in [0,1].$$

In some experiments, certain bases (e.g., all bases from one or more Solexa cycles) may be manually suppressed due to quality control. If, in a given read, a subset $J \subseteq 1..n$ of bases has been suppressed for such a reason, then the factors of the product in S^* for j in J are omitted (or, equivalently, each is replaced with 1).

Bisulfite Alignment Scores

Conversion by sodium bisulfite of unmethylated “C”s to “T”s complicates matrix comparison. The following was used for the data presented here. Assume two position–weight matrices p and q of the same length n are given, with p being from a genomic window and q being a *BS–read* (that is, a read putatively of genomic DNA processed by sodium bisulfite). A *possible bisulfite product* (PBP) of s in Σ^n is any of the $2^m \geq 1$ strings t in Σ^n obtained by independently replacing each “C” in s by either “C” or “T”, where m in $0..n$ is the number of “C”s in s. As q induces a probability distribution on Σ^n, there is a certain probability that the sequence of q is any particular such t. The expected maximum such probability over PBPs under the distribution on Σ^n induced by p is taken as the *bisulfite alignment score*, this being

$$S^*_{\mathrm{BS}}(p,q) = \sum_{s \in \Sigma^n} \Pr(s \mid p) \cdot \max_{\substack{s' \in \Sigma^n \text{ such that} \\ s' \text{ is a PBP of } s}} \Pr(\text{sequence of } q \text{ is } s')$$
$$= \sum_{s \in \Sigma^n} \left(\prod_{j=1}^{n} p_{s_j,j} \right) \cdot \max_{\substack{s' \in \Sigma^n \text{ such that} \\ s' \text{ is a PBP of } s}} \left(\prod_{j=1}^{n} q_{s'_j,j} \right)$$
$$= \sum_{s_1=1}^{4} \cdots \sum_{s_n=1}^{4} p_{s_1,1} \cdots p_{s_n,n} \begin{pmatrix} q_{s_1,1} & \text{if } s_1 \neq \text{“C”} \\ \max(q_{2,1},\, q_{4,1}) & \text{if } s_1 = \text{“C”} \end{pmatrix} \cdots \begin{pmatrix} q_{s_n,n} & \text{if } s_n \neq \text{“C”} \\ \max(q_{2,n},\, q_{4,n}) & \text{if } s_n = \text{“C”} \end{pmatrix}$$
$$= \left(\sum_{s_1=1}^{4} p_{s_1,1} \cdot \begin{pmatrix} q_{s_1,1} & \text{if } s_1 \neq 2 \\ \max(q_{2,1},\, q_{4,1}) & \text{if } s_1 = 2 \end{pmatrix} \right) \cdots \left(\sum_{s_n=1}^{4} p_{s_n,1} \cdot \begin{pmatrix} q_{s_n,n} & \text{if } s_n \neq 2 \\ \max(q_{2,n},\, q_{4,n}) & \text{if } s_n = 2 \end{pmatrix} \right)$$
$$= \prod_{j=1}^{n} \left(p_{1,j}q_{1,j} + p_{2,j} \max(q_{2,j},\, q_{4,j}) + p_{3,j}q_{3,j} + p_{4,j}q_{4,j} \right) \in [0,1].$$

Bisulfite scoring by expanding p and independently q into their probability distributions on Σ^n, further expanding each sequence from p into a uniform distribution on its PBPs, and asking for the probability of a match would have resulted in use of

$$\prod_{j=1}^{n} \left(p_{1,j}q_{1,j} + p_{2,j}\frac{q_{2,j}+q_{4,j}}{2} + p_{3,j}q_{3,j} + p_{4,j}q_{4,j} \right)$$

as a scoring formula. This formulation and others like it (e.g., without the division by 2) were rejected due to their unequal weight on letters "A", "C", "G", and "T" (e.g., scoring factors not all the same for a perfect "A", "C", "G", or "T" read basecall matching the genome perfectly) or, all else being equal, their potential for bias (favoring windows with genomic "C"s rather than "T"s, essentially due to the subset of PBPs increasing under inclusion as genomic "T"s are changed to genomic "C"s).

Statistically–Founded Mapping of Reads to Genomes

Fix a genome of interest and base length $n \geq 1$ not longer than a longest genomic sequence. Given a read q of length n (whose sequence is presumably that of a window from the genome), likelihoods — a Bayesian approach with a uniform prior on the set $W_n \neq \varnothing$ of all genomic windows of length n — are used to identify likely windows that are the origin of the sequenced portion of the fragment corresponding to the read. As w varies over W_n, $\Pr(w \mid q)$ is proportional to $\Pr(q \mid w) = S^*(p_w, q)$, where p_w is the position–weight matrix for w. Hence, the posterior distribution over w in W_n for the genomic location of q is given by

$$L_q(w) = \frac{S^*(p_w, q)}{\sum_{w' \in W_n} S^*(p_{w'}, q)} \in [0,1] \text{ (the denominator never vanishing in our applications),}$$

while $M(q) := \max\{ S^*(p_w, q) \mid w \text{ in } W_n \}$ in [0, 1] is a measure of how strongly it appears that q comes from the genome (whereas $\max\{ L_q(w) \mid w \text{ in } W_n \}$ in [0, 1], on the other hand, is one measure of how uniquely the read is localized to the genome). Note that L_q is the same for genomic windows with identical IUPAC discrete basecalls; posterior probability mass for a given window sequence independent of genomic location is uniformly distributed among genomic locations sharing that sequence.

Much effort (detailed later in this document) was expended in designing and implementing data structures and algorithms that, when given a read q, efficiently but precisely and accurately compute rigorous key information about L_q and $M(q)$. (Brute force evaluation of $S^*(p_w, q)$ for all pairs of hundreds of millions of q with hundreds of millions [for *Arabidopsis*] — or even billions [for mouse and other large genomes] — of w is presently prohibitively computationally expensive, even with SIMD vector–optimized code or exotic techniques such as GPGPU.) While a variety of information types were developed (as what is needed depends on sequencing application), in the present work reads q with $\mathrm{ESB}(q) < 20$ (taking probability "A" = "C" = "G" = "T" = 1/4 for each suppressed cycle, if any) or $M(q) < 0.01$ were rejected as inferior quality and it was determined for each read q that was not rejected all w at least 1/100 as likely as a maximum likelihood genomic location (i.e., all windows w with $S^*(p_w, q) \geq 0.01M(q)$). Read q was considered to map uniquely if and only if exactly one window resulted.

Probabilistically–Founded Mapping of BS–Reads to Genomes

With a fixed genome of interest and base length $n \geq 1$, mapping of a BS–read q of length n to the genome is generally performed similarly as to the non–bisulfite case, except with matrix comparison S^* replaced by S^*_{BS}. There are, however, some additional complications.

Conversion by sodium bisulfite generally breaks A–T, C–G basepairing (as some "C"s — paired with "G"s — are replaced with "T"s that normally basepair with "A"s) and, hence, after

further rounds of amplification via DNA replication there are generally four strands of concern: converted genomic +, converted genomic –, reverse complement to converted genomic +, and reverse complement to converted genomic –. Referring to the former two as *FW* and the latter two as *RC*, one speaks of *FW BS–reads* and *RC BS–reads*. Various strategies to deal with this ambiguity were researched; here, when working with L_q and $M(q)$ for a given read q, each window actually participated in two applications of S^*_{BS}, once with the unmodified read (to consider it as FW) and once with its reverse complement (to consider it as RC).

Assume, as is generally done in this work, that one is only concerned with reads mapping uniquely. Suppose a read q maps uniquely to window w. It is generally possible that methylation states of w other than that evidenced in q give rise to BS–reads that do not map uniquely to w, which can introduce methylation state bias in interpretations of datasets consisting of uniquely mapped reads. If there exists at least one window w' in $W_n \setminus \{w\}$ such that w and w' share at least one PBP, then w is *potentially bisulfite confusable* (PBC). As part of the offensive against methylation state bias, in the bisulfite case it was additionally required of a uniquely–mapping read that the unique window to which it maps not be PBC; such reads are said to map *BS–uniquely*. Typically, ≈1.4 million BS–unique reads were obtained per lane.

Initial Stages of Raw Image Analysis via a Subset of the Solexa Software Pipeline

Each n–cycle (here, n = 36 or 33) Illumina 1G Genome Analyzer run produces 1,004–by–1,002 pixel 16–bit grayscale raw images in TIFF format over ≈3 days, one image for each of 8 *lanes*, 200 *tiles* (a purely technical subdivision of lanes), 4 *colors* from which "A", "C", "G", and "T" basecall information is eventually derived, and n *cycles* (bases) for a total of 6,400 n images (here, 230,400 or 211,200). As each image is 2,012,138 bytes, each run of 8 lanes produces ~395 to 432 GiB of raw data. Each lane can be loaded with a DNA library independently of other lanes and flow cells containing lanes used in this study contained lanes loaded with other libraries as well.

A subset of version 0.2.2.4 of the Solexa software pipeline was used to perform the initial stages of raw image analysis described in this and the next paragraph. Images of different colors are registered (*offsets* are computed and effectively applied) so that afterwards a given (x, y) pixel location for a given tile in a given lane ideally corresponds to the same physical flow cell location across colors and cycles. *Spot calling* is performed, grouping pixel locations per tile per lane so that ideally the pixels in each group correspond to a solid–phase–amplified DNA *cluster* of molecules on the flow cell and groups enumerate such clusters; each called spot becomes one read. Signal intensity relative to background for each color for each cycle for each spot is integrated to obtain a four–tuple $I := (I_A, I_C, I_G, I_T)$ of real *intensities* (`_int` intensities) per cycle per spot. The number of clusters is related to the concentration of prepared DNA fragments loaded onto the flow cell and is also affected by inherent variation across flow cells and the preparation and sequencing processes. Owing to the random placement of clusters, flow cell surface area is wasted (but intensities are cleaner) when there are few clusters, but clusters tend to overlap if there are too many. It was found that ≈17,000 called spots per tile via loading prepared DNA at 1 to 2 picomolar concentration was an effective compromise, with many runs of the present work being in the vicinity of this density. Rather than using some threshold (e.g., 0.6) on the Solexa software pipeline's "chastity" measure (as computed by its QUAHOG component and recorded in the `_qhg` files) to remove spots it believes to be contaminated by overlap, all spots are allowed to proceed to the more stringent and presumably less biased (by the statistics of the mapping process in regard to an approximate model of reads, overlaps of them, and observations of chastity behavior) selection process of mapping to a genome.

Due to the overall system response of the cleavable–fluorophore–containing reversible–terminating nucleotides, lasers, optical filters, light path components, and camera of the Solexa instrument as well as the kinetics of the chemistry of the Solexa sequencing process, the components of intensity vectors I are not responsive to purely one distinct base each. Hence, a *color crosstalk matrix* is estimated and its inverse applied so that ideally the ith component of corrected intensity vectors responds purely to "A", "C", "G", or "T" for $i = 1, 2, 3$, and 4, respectively. Also at this time, a *phasing* correction is estimated and applied in an attempt to account for temporal de–synchronization of the population of molecules forming each cluster (with some molecules running fast due to, e.g., incorporation of more than one base per cycle, while others run slow due to, e.g., failure of removal of one or more terminators). The result is a four–dimensional $S(r, j) := (S_A(r, j), S_C(r, j), S_G(r, j), S_T(r, j))$ *signal vector* (`_sig2` signals) for each cycle j in $1..n$ of every read r. These signal vectors are the primary input data used to perform *basecalling*, which is understood here to be the formation of a position–weight matrix for each read. Note that due to use in this work of fitted Gaussian mixture models for basecalling (see below), in the context of the entire analysis the corrections at this stage are only initial corrections to which further finely–tuned corrections are effectively applied to each individual cycle of each individual lane.

Certain components of version 0.2.2.4 of the Solexa software pipeline require a roughly balanced mixture of "A"s, "C"s, "G"s, and "T"s for the *autocalibration* (as it is known) of the color crosstalk matrix and phasing. As genomic "C"s tend to be rather unmethylated at the level of the whole *Arabidopsis* genome and the genome starts as being poor in C and G (ATH1 is ~36% C + G), the percentage of "C"s after bisulfite treatment is very low, although this is ameliorated somewhat by the presence of RC BS–reads. Hence, each flow cell contained a non–bisulfite (e.g., an untreated whole genome) library used for autocalibration (and possibly other non–bisulfite libraries as well).

In some runs, weak signals were observed for one or more colors in the first cycles of a run. As version 0.2.2.4 of the Solexa software pipeline focuses on the first cycle given to it for determination of numerous parameters, in these cases such cycles were manually excluded. Further, due to the biochemical construction of the bisulfite DNA libraries, the first five cycles are generally those that sequence bases not derived from the original organismal DNA of interest. Hence, subsequent analysis focuses on the tail $31 = 36 – 5$ or $28 = 33 – 5$ cycles.

Basecalling via Gaussian Mixture Models

As generally many fewer than $n \approx 30$ bases are required to have only a comparatively small fraction of DNA sequences of length n appear in multiple windows or be PBC (even for large genomes, such as human), mapping of reads and BS–reads to reference genomes are typically somewhat robust processes that do not demand exceptionally high accuracy of absolutely every single base for a given read. However, for estimation of methylation at single–base resolution, it is critical that individual bases be called with high accuracy. Due to the extreme depletion of "C"s after bisulfite conversion (and, e.g., the generally low level of CHH methylation if any at any given site), error rates even in the vicinity of a single percent may be quite misleading as to final estimated methylation.

The apparent statistical well–behavedness of the distribution of signal vectors S within cycles of a given lane and the systematic drifts and changes of these distributions across cycles of the lane (apparently due in part to non–constancy of crosstalk and phasing across lanes and cycles)

suggested that explicit modeling of signal vector distributions would be fruitful. This was confirmed after the development of basecalling via fitted multidimensional Gaussian mixture models outlined in one form presently, the application of which has been observed, e.g., to provide significant reduction in post–mapping occurrences of alignments of genomic bases to read basecalls not consistent with bisulfite conversion.

Full multi–component four–dimensional Gaussian mixtures (without constraints such as diagonal covariances) are fitted by braked expectation maximization (EM) to signal vectors S on a per–lane, per–cycle basis. EM is an iterative local optimization method that requires a fairly close initial approximation to an optimal mixture to converge to it and so fitted models must generally be monitored for reasonability as regards certain application–driven desiderata (and also as suboptimal fits that are more reasonable than optimal fits may be preferred). Starting at cycle 6, the color crosstalk correction applied by the Solexa software pipeline serves to normalize S vectors sufficiently that it is often not difficult to form an automated suitable initial mixture model for that cycle, and the fitted mixture for each cycle generally serves as a suitable initial model for the next cycle. In fitting each mixture, up to 50 EM iterations are performed, with a provision made for early termination when all parameters of interest change sufficiently slightly on successive iterations.

For a typical cycle of a typical lane, the S vector density concentrates in four distinct and roughly hyperellipsoidal fuzzy regions with different means, radii, principal axes, and overall proportion (the overall proportions being different due to the uneven distribution of post–bisulfite bases). These concentrations of signal vectors correspond to “A”, “C”, “G”, and “T”. However, due to physical overlap on the flow cell of some of the clusters corresponding to spots, there are also fuzzy sheets of density between pairs of these hyperellipsoids, most pronounced between the “A” and “T” concentrations due to their high proportion; fitting must generally be adapted to account for these. Additional considerations are made for S whose spot (x, y) pixel location to the nearest other spot is unusually small or large, for S in regions of low density of signal vectors, and for S that are particularly far away from or close to the origin. The ANN nearest–neighbors C++ library (http://www.cs.umd.edu/~mount/ANN/) of David Mount and Sunil Arya and the E and M steps of Tim Bailey’s Gaussian Mixture Model and Gaussian Kernel MATLAB Utilities (http://www-personal.acfr.usyd.edu.au/tbailey/software/gmm_utilities.htm) are incorporated.

Desiderata mentioned in the paragraph just before the previous one are a clear association of “A”, “C”, “G”, and “T” to distinct mixture components as judged by angle between mean vectors and canonical axes, mixture probabilities within expected intervals, and bounds on ratios of hyperellipsoid volumes, of hyperellipsoid maximum radii, and of largest to second largest radii of individual hyperellipsoids. While automated monitoring is necessary to efficiently deal with the massive volume of raw data encompassed by a project of this size, such can only go so far as tendency for variation in physical materials (flow cells, reagents, …), physical conditions, and chemistry as well as hardware glitches and failures results in a wide and hard–to–anticipate variety of observed anomalies in the terapixels of collected images. Hence, a variety of two–dimensional projections of empirical S density and fitted Gaussian mixture components are arrayed into frames of a losslessly–compressed Apple QuickTime movie, one frame per cycle and one movie per lane. These movies provide a rapid means of manual evaluation of both the quality of a lane and the reasonability of the final mixture fits. Cycles that display any indication of an experimental mishap or for which the fitted mixtures are not manifestly highly co–located with empirical density are suppressed (hence the final paragraph under *Comparison of Matrices* above). Lanes which would otherwise have had more than four suppressed cycles were completely omitted from the study.

For a given cycle of a given lane, basecalling is straightforward once a Gaussian mixture has been fit for it. For m = 1, 2, 3, and 4 (corresponding to "A", "C", "G", and "T", respectively), let real $p^{(m)} > 0$ be the mixture probability for mixture component m (with $p^{(1)} + p^{(2)} + p^{(3)} + p^{(4)} = 1$), let $\mu^{(m)}$ be the real 4–by–1 column vector giving the mean of the mth component, and let $\Sigma^{(m)}$ be the 4–by–4 real symmetric positive definite covariance matrix of mixture component m. Then

$$\Pr(S_A, S_C, S_G, S_T \mid m) = \Pr(S \mid m) = \frac{\exp\left(-\frac{1}{2}(S-\mu^{(m)})^T (\text{inverse}\,\Sigma^{(m)})\,(S-\mu^{(m)})\right)}{4\pi^2\sqrt{\text{determinant}\,\Sigma^{(m)}}} > 0$$

for all S in $\mathbb{R}^4$ is the Gaussian density of mixture component m. Hence, by Bayes' Theorem, $\Pr(m \mid S)$ is proportional to $\Pr(m)\ \Pr(S \mid m) = p^{(m)}\ \Pr(S \mid m)$ as m varies in 1..4, so that the basecall (before a few adjustments described below are applied) for a given signal vector S is the column vector

$$\begin{aligned} b(S) = {} & \left(\Pr(\text{"A"} \mid S),\ \Pr(\text{"C"} \mid S),\ \Pr(\text{"G"} \mid S),\ \Pr(\text{"T"} \mid S)\right) = \\ & \left(\sum_{m=1}^{4} p^{(m)} \frac{\exp\left(-\frac{1}{2}(S-\mu^{(m)})^T (\text{inverse}\,\Sigma^{(m)})\,(S-\mu^{(m)})\right)}{\sqrt{\text{determinant}\,\Sigma^{(m)}}}\right)^{-1} \cdot \\ & \left(p^{(1)} \exp\left(-\tfrac{1}{2}(S-\mu^{(1)})^T (\text{inverse}\,\Sigma^{(1)})\,(S-\mu^{(1)})\right) \Big/ \sqrt{\text{determinant}\,\Sigma^{(1)}}\,,\right. \\ & \quad p^{(2)} \exp\left(-\tfrac{1}{2}(S-\mu^{(2)})^T (\text{inverse}\,\Sigma^{(2)})\,(S-\mu^{(2)})\right) \Big/ \sqrt{\text{determinant}\,\Sigma^{(2)}}\,, \\ & \quad p^{(3)} \exp\left(-\tfrac{1}{2}(S-\mu^{(3)})^T (\text{inverse}\,\Sigma^{(3)})\,(S-\mu^{(3)})\right) \Big/ \sqrt{\text{determinant}\,\Sigma^{(3)}}\,, \\ & \quad \left. p^{(4)} \exp\left(-\tfrac{1}{2}(S-\mu^{(4)})^T (\text{inverse}\,\Sigma^{(4)})\,(S-\mu^{(4)})\right) \Big/ \sqrt{\text{determinant}\,\Sigma^{(4)}}\right). \end{aligned}$$

Hence, the position–weight matrix for a read r is

$$B(r) = \begin{bmatrix} b(S(r,1))_1 & b(S(r,2))_1 & \cdots & b(S(r,n))_1 \\ b(S(r,1))_2 & b(S(r,2))_2 & \cdots & b(S(r,n))_2 \\ b(S(r,1))_3 & b(S(r,2))_3 & \cdots & b(S(r,n))_3 \\ b(S(r,1))_4 & b(S(r,2))_4 & \cdots & b(S(r,n))_4 \end{bmatrix}.$$

It is, however, desirable to not call bases whose signal vector is unusually distant from the means of all four of the "A", "C", "G", and "T" mixture components. Let $Q(S)$ be the four–dimensional real vector with successive entries in (0, 1] being the outer Gaussian quantile of S in the respective component of the mixture model. That is, let $Q(S)_m$ for m in 1..4 be the probability that S' is at most as likely as S (i.e., $\Pr(S' \mid m) \le \Pr(S \mid m)$) when S' is an independent multinormal random variable of mean $\mu^{(m)}$ and covariance $\Sigma^{(m)}$. If all four entries of $Q(S)$ are below one billionth, then S is unlikely to be from the model (it is likely anomalous in some experimental or analytic way) and the basecall $b(S)$ for it was replaced with (1/4, 1/4, 1/4, 1/4).

In addition to addressing cases where a mixture model might otherwise be extrapolated too far out in signal vector space, one is also concerned with the model assigning basecalls that are overly precise (i.e., placing all but an extremely small amount of the probability mass on just one of "A", "C", "G", or "T"). Some errors are introduced even before the Solexa measurement process (e.g., DNA polymerases used in DNA library construction do not have perfect fidelity

and a base error at such a stage can amplify to an entire cluster on a flow cell). Further, it was desired to continue an overall conservative tendency in the present study. Hence, if a basecall $b(S)$ arises that has an entry in excess of 0.98 (if so, there is only one such entry), then the excess was equidistributed to the other three entries.

For each lane, $p^{(m)}$ for m in 1..4 were fixed across cycles via re–normalized medians. Additionally, analyses in the present work that are downstream of mapping operate on discretized single–letter basecalls. A position–weight matrix was converted by replacing each column that has an entry of 0.9 or higher (there can be at most one such entry) with the single letter that corresponds to its row (be it "A", "C", "G", or "T"). Each remaining column was replaced with "N". Degree of methylation at a given genomic "C" was then determined from tallies of the number of manifest read "C" bases versus manifest read "T" bases mapping to that genomic "C" (taking reverse complements as appropriate for BS–reads deemed RC).

Non–Conversion Filtration

It is well–known from low–throughput sequencing of molecular clones of DNA treated with sodium bisulfite that *non–conversion* can occur, that is, some genomic "C" bases remain "C" after treatment regardless of their methylation state. In low–throughput traditional bisulfite sequencing, these are typically filtered out manually; if such occurrences are not identified, spurious apparent methylation results. Non–conversion usually arises because of short stretches of non–denatured DNA (as sodium bisulfite operates on single–stranded DNA) with the result that it appears as several adjacent unconverted cytosines [Warnecke P. M. et al., Methods. 2002 2:101–7].

Several instances of apparently unconverted sequences were discovered via manual inspection of a variety of regions of an unfiltered version of the dataset as displayed in the UCSC genome browser (see below). These instances were small dense patches of "methylation" found in regions that would not be expected to be methylated, such as the promoters of active genes and the unmethylated chloroplast genome. Often, this "methylation" derived from reads in which all genomic "C"s appear as "C"s in the read, either alone or in small groups, and thus appear to be unconverted. To confirm this, four of these regions were validated by traditional bisulfite sequencing, and *all* were completely unmethylated (shown in Supplementary Figure 17). Looking carefully at the patterns, it was determined that eliminating reads with three CHH sites in a row virtually eliminated these unconverted reads and this was the largest (i.e., most selective) number of adjacent CHH sites needed for this filter to be effective.

The total fraction of reads removed is ~0.23%. While this filter undoubtedly removes some genuine methylation, based on the expected frequency of CHH methylation at individual sites (roughly 10% at most loci; see Supplementary Figure 1) and prior experience of the Jacobsen lab (e.g., the observation that methylation at sites of even close proximity is rather independent within individual clones across clones covering a given small genomic region), this filter is expected to remove only a small amount of veridical methylation.

A precise description of the filter is as follows. Given a read and a mapped genomic location for it with discrete single–letter IUPAC genomic bases g and discrete read bases r (with r already reverse complemented in the RC case), consider only the "C"s in g, retaining their order and classifying the genomic context of each (as each sits in the entire genomic sequence in question) as CG, CHG, or CHH. For each, consider it methylated if and only if the corresponding base in r

is "C" (and not part of a suppressed cycle). The read is rejected by the filter if and only if there are three consecutive methylated CHHs.

Presentation in the UCSC Genome Browser

From prior work, a local mirror (http://epigenomics.mcdb.ucla.edu/) of the UCSC Genome Browser (http://genome.ucsc.edu/) that had already been augmented with the *Arabidopsis* genome was available. While the browser was not designed for datasets on the scale of whole genome single–base DNA methylation, it is serviceable providing that the track loading tools (which operate with the browser's formats [which are space–inefficient in this application] and assume the underlying data of whole tracks can be read entirely into memory at one time) are bypassed and MySQL tables in the browser's internal formats are filled directly.

The only table columns for BED–style tracks that cause any difficulty are *bin* (derived from *chromStart* and *chromEnd*), which serves as the browser's device to improve efficiency of spatial searching (e.g., to retrieve all objects of a track intersecting a given end–user's display), and *reserved* (derived from external *itemRgb*), which encodes color. The other fields are essentially direct from the external BED format of http://genome.ucsc.edu/FAQ/FAQformat already documented on the UCSC site. Needed details were derived from direct examination of the UCSC source code (obtained via CVS; see http://genome.ucsc.edu/admin/cvs.html).

Assuming 32–bit two's complement `int`s, *bin* may be computed by the C++ function

```
int binFromRange(int chromStart, int chromEnd) {
    chromStart >>= 17;  chromEnd = ((chromEnd-1) >> 17);
    if(chromStart == chromEnd)  return(chromStart + 585);

    chromStart >>=  3;  chromEnd >>= 3;
    if(chromStart == chromEnd)  return(chromStart +  73);

    chromStart >>=  3;  chromEnd >>= 3;
    if(chromStart == chromEnd)  return(chromStart +   9);

    chromStart >>=  3;  chromEnd >>= 3;
    if(chromStart == chromEnd)  return(chromStart +   1);

    chromStart >>=  3;  chromEnd >>= 3;
    if(chromStart == chromEnd)  return(chromStart +   0);

    /*FAILURE: [chromStart, chromEnd) out of bin-able range*/
    std::abort() /*...from #include <cstdlib>*/;
    }
```

and, if the three comma–separated fields in 0..255 of *itemRgb* are *red*, *green*, and *blue*, respectively, *reserved* may be computed by the C++ function

```
unsigned int bedParseRgb(const unsigned int red,
const unsigned int green, const unsigned int blue)
{ return((red<<16) | (green<<8) | blue); }
```

(trivial transliterations of these two functions also work in, e.g., Perl).

As the MySQL MyISAM engine fills the backing store file of a table in the same order as rows arrive via bulk inserts (performed via, e.g., `LOAD DATA INFILE`), it is helpful to sort rows by genomic position before loading. GNU `sort`, in particular, has no trouble with text files even of hundreds of gigabytes on hardware of today. (Indeed, the desire to sort is perhaps the main reason the UCSC loading tools start by loading entire tracks into memory. GNU `sort` scales much better as it transitions to strategies designed for external sorting and provides the `--buffer-size=` command line option for configuring the size of its in–memory buffer.) Further, use of `myisampack` is helpful to overcome some inefficiencies of the UCSC internal table formats in the present application.

Tracks that display individual read manifest "C"s and "T"s (colored according to methylation state and genomic context) as they align to genomic "C"s are straightforward. It is more difficult to choose how to display reads as there is no obvious track type choice that can display short and generally overlapping objects aligned to the genome with arbitrary patterns of trinary state ("methylated", "unmethylated", and "non–genomic–C") within each at single–base resolution. However, prepending and suffixing single–base "UTRs" permits display of arbitrary patterns of binary state via "exon"–"intron" distinctions, and advantage was taken of this to exhibit tracks that display reads with "methylated–genomic–C" vs. "unmethylated–or–non–genomic–C" visually apparent for every base of every displayed read when zoomed in sufficiently close.

Determination of Methylation Status for Cytosines

Once reads meeting all criteria are mapped to the genome, one may tabulate at each genomic "C" the number $a \geq 0$ of manifest read–"C"s and $b \geq 0$ of manifest read–"T"s mapping there. Though ~93% of theoretically–coverable cytosines were covered with at least one sequenced read, most downstream analyses were restricted to genomic "C"s having $a + b \geq 5$ to reduce the error in percent methylation estimates inherent in the statistics of few counts. Methylation at each position was estimated as the fraction $a / (a + b)$.

Additional Details of Data Analysis

All Arabidopsis data analyzed in the paper are from BS–unique sequences, except for the rDNA and telomere analyses in which all sequences mapping to these regions were analyzed. For the basic analysis (not using mapping) of overall methylation in the wild type and np95–/– mouse libraries, the stock Solexa pipeline was used — including a "chastity" ≥ 0.6 filter to eliminate sequences of presumably poor quality — after which the number of apparent CpG (as exhibited in the Solexa `_seq` stock discrete basecalls) remaining in the reads were tallied. Hence, all passing sequences were included in this case regardless of their uniqueness in the genome or whether they were present in the assembled portions of the mouse genome.

Algorithmic and Implementational Details of Cokus Pipeline

Overview

The key to CokusAlignment fast mapping of hundreds of millions of short, imprecise reads to a reference genome of hundreds of millions to billions of well–known nucleotides while making use of the full probabilistic content of reads is the ability to rapidly but rigorously eliminate large fractions of W_n — the set of all genomic windows of length n — as candidates

for highest–scoring windows for each read. This is done by pre–processing the reference genome into an efficient encoding of a tree structure invented for this work and intelligently traversing it using branch–and–bound techniques via a double–ended priority queue. A sequence of simplified data structures and algorithms (some of which are not practical, but are easier to explain) will be presented, leading up to the data structures and algorithms actually employed by the highly–optimized C++ versions of CokusAlignment used in this work.

Key Ideas

As scoring against a given window w in W_n is a function of the window sequence of w and not the genomic location of w, it simplifies mapping to consider instead of W_n the set

$$G_n := \{ \text{the IUPAC nucleotide sequence of } w \mid w \in W_n \}$$

of distinct *window sequences*, and the *location whole–genome copy number*

$$\mathrm{WGCN_L}(g) := \#\{ w \in W_n \mid \text{the IUPAC nucleotide sequence of } w \text{ is } g \} \geq 1$$

and a fixed but arbitrary *representative location*

$$\mathrm{RL}(g) \in \{ w \in W_n \mid \text{the IUPAC nucleotide sequence of } w \text{ is } g \} \neq \varnothing$$

of each g in G_n. For typical genomes (even large ones, e.g., mouse) and n, the location whole–genome copy numbers of all but a fairly small percentage of g in G_n are 1. Put

$$G_n^* := \bigcup_{i=0}^{n} \{ \text{prefix of length } i \text{ of } g \mid g \in G_n \},$$

the set of all prefixes of G_n, and form the graph T whose set of vertices is G_n^* and that has a directed edge from node x to node y if and only if x is a prefix of y and exactly one letter shorter. Label each leaf g with $\mathrm{WGCN_L}(g)$ and $\mathrm{RL}(g)$, and label each edge (x, y) with the single IUPAC letter that ends y. T is a (typically enormous) rooted, directed tree with ε, the empty string, as root.

Take q to be the position–weight matrix of the current read for which mapping is desired, and put $S'(x) := S^*(x, \text{prefix of } q \text{ of same length as } x)$ or $S_{\mathrm{BS}}^*(x, \text{prefix of } q \text{ of same length as } x)$ as appropriate for all $x \in \Sigma^0 \cup \cdots \cup \Sigma^n$ (considering x as if it were from a genomic sequence). The primary operation of mapping q is visitation of the leaves of T in an order such that S' is non–increasing; this primitive and variations of it suffice to answer many questions about L_q and $M(q)$ (or the bisulfite equivalents, if appropriate; in the following, L and M will be used in this way without further comment). Note that $S'(\varepsilon) = 1$ and $S'(y)$ is easily computed from $S'(x)$ by multiplying by a simple factor when y extends x by a single letter (that is, when (x, y) is an edge; the factor is a function of the label on this edge and q). Note that the factor is in $[0, 1]$ and, hence, $S'(y) \leq S'(x)$. The basic idea is to compute S' at nodes of the tree, starting at the root and flowing from parents to children to eventually reach leaves, but doing so in a prioritized way so that, in the end, in typical applications most q avoid visiting almost all of the tree.

Let D be an initially empty ordinary priority queue of nodes with priority corresponding to S'. Hence, if D is non–empty, a node with maximal S' among the nodes in D is available at the top of D. Enqueue ε with $S'(\varepsilon) = 1$. While D is not empty, dequeue the top of D as x and *visit* x,

which for the moment is the computation of $S'(y)$ and enqueueing of y for each child y of x (if any). This procedure achieves the basic goal; in fact, all nodes (both leaves and non–leaves) are visited in an order such that S' decreases (with ties visited consecutively in an as–yet–arbitrary order) so that, in particular, the subsequence of visits to leaves are such that S' is non–increasing. Note that, for example, $M(q)$ (and, using RL, a window achieving it) can be determined from the first leaf visited.

Many refinements are required to make the basic traversal of the previous paragraph practical. These are both technical (e.g., traditional representations of trees in computer memory are too inefficient in this application and need replacing) that improve space and time consumption by constant (but still critical) factors, and algorithmic (e.g., dynamic prunings of T) that can provide more dramatic savings.

Algorithmic Optimizations for Practical Execution

Suppose one is only interested in g in G_n such that $S'(g) \geq \alpha$ for some real parameter α (e.g., 0.01). Then one does not need to visit any node x such that $S'(x) < \alpha$ nor (and this is key) *any of the descendents y of x* since $S'(y) \leq S'(x) < \alpha$. Hence, a first refinement to the basic traversal is to not enqueue any node whose S' is below α. Such *pruning* (that is, non–visitation of a node and its descendents, by the criteria introduced in this and later paragraphs) is the major means by which CokusAlignment is greatly sped up relative to brute force implementations. For example, pruning of a node not far below the root can safely (and nearly instantaneously) eliminate many millions of genomic windows from consideration.

Next, suppose real parameter β in [0, 1] is specified (e.g., 0.01) and one is only interested in those g such that $S'(g) \geq \beta M(q)$ (or in determining that there is no g in G_n such that $S'(g) \geq \alpha$). One may proceed as in the previous paragraph, initially pruning nodes x such that $S'(x) < \beta \alpha$ until the first leaf is visited, establishing $M(q)$. (If the queue empties before visiting any leaves or if S' for the top of D drops below α, then there is no g in G_n such that $S'(g) \geq \alpha$.) Processing then continues until the queue empties, except pruning nodes x such that $S'(x) < \beta M(q)$; the desired leaves are exactly those visited.

It is apparent from the previous two paragraphs that, in general, the sooner a high pruning level on S' can be established, the better. While the basecalls in q can be sufficiently fuzzy that taking the most likely call (be it "A", "C", "G", or "T") at each base of q may not be the window sequence of a high–scoring window, it is not uncommon for this to be the case. Hence, while not sound if used in isolation, a greedy approach may be employed before the queue–based main loop to provide a lower bound on $M(q)$ that may result in more extensive and earlier pruning (but still safe — i.e., mapping speed is generally improved without ever sacrificing accuracy) than would otherwise be had. Start at ε and continue until a leaf x is reached; at non–leaves, follow an edge that leads to a node of highest S' (breaking ties arbitrarily). Then proceed as before, but as $M(q) \geq S'(x)$, if the pruning level at any given moment is below $\beta S'(x)$, then it can be taken as $\beta S'(x)$ instead.

In the next refinement, S' is permitted to increase as nodes are visited. However, as mentioned previously, non–increasingness of S' is required on the subsequence of visits to leaves, and this will be maintained. Until now, when prioritizing nodes and pruning, non–leaves x generally could be viewed as having higher scores than strictly required in that $S'(y)$ is often rather smaller than $S'(x)$ for all children y of x that are leaves: the columns of q and the probabilities associated with genomic bases are generally smaller than 1, and hence the factors

yet to be multiplied to $S'(x)$ to obtain the $S'(y)$ are generally smaller than 1 as well. One can form *tail bounds* that provide conservative (but generally tighter than heretofore) upper bounds on S' of leaf–descendents of general nodes as follows. Put $\Xi := \{A, B, C, D, G, H, K, M, N, R, S, T, V, W, Y\}$, the IUPAC nucleotide alphabet. Identify each member of Ξ with the probability distribution on ("A", "C", "G", "T") associated to it as a base from a genomic sequence. If x is a node of length m in $0..n$ and y is a descendent of x that is a leaf, then in the non–bisulfite case,

$$\begin{aligned} S'(y) &\le \max_{c \in \Xi^{(m+1)..n}} S'(\text{concatenation of } x \text{ and } c) \\ &= \max_{c \in \Xi^{(m+1)..n}} \left(S'(x) \prod_{j=m+1}^{n} \left((c_j)_1 q_{1,j} + (c_j)_2 q_{2,j} + (c_j)_3 q_{3,j} + (c_j)_4 q_{4,j} \right) \right) \\ &= S'(x) \cdot \underbrace{\prod_{j=m+1}^{n} \max_{c \in \Xi} \left(c_1 q_{1,j} + c_2 q_{2,j} + c_3 q_{3,j} + c_4 q_{4,j} \right)}_{=: \lambda(m) \in [0,1],\ \text{the } q \text{ tail bound for leaf-extensions of length } m \text{ nodes}}, \end{aligned}$$

and in the bisulfite case,

$$\begin{aligned} S'(y) &\le \max_{c \in \Xi^{(m+1)..n}} S'(\text{concatenation of } x \text{ and } c) \\ &= \max_{c \in \Xi^{(m+1)..n}} \left(S'(x) \prod_{j=m+1}^{n} \left((c_j)_1 q_{1,j} + (c_j)_2 \max(q_{2,j}, q_{4,j}) + (c_j)_3 q_{3,j} + (c_j)_4 q_{4,j} \right) \right) \\ &= S'(x) \cdot \underbrace{\prod_{j=m+1}^{n} \max_{c \in \Xi} \left(c_1 q_{1,j} + c_2 \max(q_{2,j}, q_{4,j}) + c_3 q_{3,j} + c_4 q_{4,j} \right)}_{=: \lambda(m) \in [0,1],\ \text{the } q \text{ tail bound for leaf-extensions of length } m \text{ nodes}}. \end{aligned}$$

Hence (in either case), an upper bound for $S'(y)$ for all leaf–descendents y of any given node x of T is $S''(x) := S'(x)\ \lambda(\text{length } x) \le S'(x)$. Priority queue D may now be ordered and pruning may now be triggered by S'' instead of S'. Also, one can conclude that there is no g in G_n such that $S'(g) \ge \alpha$ as soon as S'' (rather than S') for the top of D drops below α.

It is convenient now to comment on how two particular supported features are handled. First, as discussed earlier, it is useful to be able to inhibit a manually–specified subset of cycles from scoring of reads. (Suppressed subsets are typically small, e.g., size 0..4, as suppression of larger subsets degrades the efficiencies otherwise made possible by T and then the means to be described momentarily of handling suppressed bases is not so effective. In such cases, if there are not too many suppression patterns, then a new, generally "discontinuous" T customized for each particular pattern of bases suppressed can be constructed; this was not needed in the present work and is not elaborated here.) If cycle j' in $1..n$ is among those suppressed, then when computing $S'(x')$ for a node x' of length j' from its prefix x of length $j' - 1$, the multiplicative factor is taken to be 1 so that $S'(x') = S'(x)$ (and S' of all descendents of x' are consequently automatically corrected as well). To maintain S'', the $j = j'$ factors in the products defining λ are also omitted. There is an interaction between suppressed cycles and greedy establishment of lower bounds introduced above; this is discussed when timings for *Arabidopsis* are given later.

Second, in the bisulfite case, it may be desired to permit a read to be FW, RC, or both. It is straightforward to permit just one of FW and RC. While it is simple to handle allowance of both FW and RC by two separate, independent traversals of T (namely, once with q and once with its reverse complement), this is not generally as efficient as the means to be described presently;

typically one of these designations (FW or RC) produces no high–scoring window and to determine such can be relatively expensive compared to the other designation in which traversal often provides a positive result by reaching at least one leaf. The idea is to marry both traversals, permitting the fast designation to rapidly reach positive results and enable good pruning as pruning in either designation applies to both designations (so that the highest pruning in either is applicable at any given moment). Each entry in priority queue D is extended with a Boolean flag that indicates whether it is part of the FW traversal or the RC traversal. The greedy lower bound performed before the main loop can be taken to be the maximum of that for FW and that for RC. Enqueueing of ε is attempted twice at the beginning of the main loop, once for the start of the FW traversal and once for the start of the RC traversal. The main loop then proceeds as before, except switching between working on the FW or RC traversal moment–to–moment according to the flag of the top element of D. If one wishes to provide a Bayesian prior with Pr(FW) = γ in (0, 1) and Pr(RC) = 1 – γ, this is easily done (although this was not used in the present work) by prefixing Pr(FW) or Pr(RC) as appropriate for the current traversal as a multiplicative factor to S'; λ is unchanged and S'' is automatically adjusted via S'.

The final algorithmic refinement employed is the use of a double–ended priority queue for D. With an ordinary priority queue, when the pruning level is raised, the "bottom" elements in D whose S'' is below the new level remain in D for a relatively long time (until all elements — both existing and yet–to–be–inserted — above them are processed). These bottom elements are useless in that as soon as one rises to the top of the queue, mapping for q is finished just as it would be if D were to become empty at that moment instead. Bottom elements cause all operations on D involving non–bottom elements to be slower than need be. For example, an efficient implementation of an ordinary priority queue of size m is via an array–based heap, with $O(1)$ time for retrieval of the top of the queue and $O(\log m)$ time to enqueue a new element or dequeue the top of the queue, and $\log m$ is larger than need be if there are bottom elements. However, by using two array–based heaps (or interleaving them, as in the versions of CokusAlignment presented here) and cross–indexing them, it is possible to provide a *double–ended priority queue* of size m that provides $O(1)$ time for retrieval of a top (i.e., highest priority) *and simultaneously bottom* (i.e., lowest priority) element of the queue and still consume only $O(\log m)$ time to enqueue a new element or dequeue the top *or bottom* of the queue. Using such a double–ended priority queue for D enables all elements whose S'' is below the new pruning level to be jettisoned from D when the pruning level is raised.

Computational Optimizations for Practical Execution

With description of the algorithmic refinements complete, details of the implementations that result in only constant factors of efficiency improvement are now discussed. These details are, however, still essential to achieve practicality on problems of interest on reasonable hardware. The first, and most important, is an efficient binary encoding of T. To illustrate the necessity, a traditional in–memory form of T might use 15 link pointers (one for each element of Ξ as possible edges, with null pointers for non–appearing labels) for each non–leaf node. In this case, 64–bit pointers would be required for even smaller genomes (such as *Arabidopsis*), resulting in 120 bytes of memory under heavy random access per non–leaf. As T for both strands of ATH1 chr1..5/C/M (366,923 bp mitochondrion) with n = 31 already has 3,782,356,392 non–leaves, a completely unreasonable amount of RAM (in excess of 422 GiB) would be required.

Initially, as *Arabidopsis* was the only target reference genome, binary encodings optimized for it ("CokusV8A") were employed; these are described first. Later, as use of larger genomes (such as human or mouse) was desired, encodings ("CokusV8B") that are slightly less efficient for smaller genomes but that are capable of capturing large genomes were implemented, and

these are described second. Should even larger genomes (or meta–genomes) be required, further encodings are easily implemented.

While members of Ξ other than A, C, G, T, and N do occur in reference genomes, these five are typically the most used letters in genomic sequences by far. Hence, it is generally wasteful to reserve links for all 15 members of Ξ for every node; given that presence/absence of an edge with a given label can be encoded with a single bit (that controls a much larger, say, 32 to 64 bits of actual link data), it is helpful to employ nodes of variable size that only consume space for links actually present. Thus, non–leaf nodes come in two varieties: "short" and "long". Both are sequences of binary bits packed into sequences of 8–bit bytes from most significant bit (msb) to least significant bit (lsb) within each byte. A short node is a 0 bit (to indicate shortness) followed by 7 flag bits that indicate presence (1) or absence (0) of 1..7 links: N, S, W, T, G, C, and A (in order), followed by actual links. A long node is a 1 bit (to indicate longness) followed by 15 flag bits that indicate presence (1) or absence (0) of 1..15 links: N, S, W, T, G, C, A, K, Y, R, M, B, D, H, and V (in order), followed by actual links.

Links are presented sequentially in the order N, S, W, T, G, C, A, K, Y, R, M, B, D, H, and finally V, with absent labels skipped. Each link is either a link to another non–leaf, a "short" link to a leaf, or a "long" link to a leaf. Links are encoded as either one or two 32–bit little–endian quantities. A link to a non–leaf is encoded as a 0 bit (indicating link to a non–leaf) followed by 31 bits that give the zero–based offset in bytes from the start of the binary data encoding the entirety of T to the target non–leaf. A long link to a leaf is given by two consecutive 1 bits (to indicate a long link), followed by 30 bits specifying which genomic sequence and the position on that sequence for the window designated as the representative location for the target leaf, followed by an unsigned 32–bit binary integer giving the location whole–genome copy number of the target leaf. Since location whole–genome copy numbers of 1 are typically common, space can be saved by using short links, which start with a 1 bit followed by a 0 bit and omit the 32–bit location whole–genome copy number (implicitly taking it to be 1), but are otherwise the same as long links.

The precise 30–bit encoding of which genomic sequence and the position on that sequence for the windows associated with representative locations is not particularly important for the present discussion, but is included here for the sake of completeness. The first three bits give an unsigned binary integer that specify the *Arabidopsis* chromosome (1..7, with 6 = chloroplast and 7 = mitochondrion), followed by a flag (0 = plus, 1 = minus) giving the strand, followed by a 26–bit unsigned binary integer x. Index the bases on the plus strands of chromosomes increasing starting from zero in the 5′ to 3′ direction. The index associated with a minus strand base is that of the plus strand base paired to it. Then x gives the index of the 5′–most base of the window.

As already mentioned, T as described heretofore typically has an impractically large number of nodes (e.g., for $n = 31$ for *Arabidopsis*, 3,782,356,392 non–leaves and 223,891,022 leaves). However, the general shape of typical trees suggests many nodes are not truly required. While all or nearly all members of Σ^m for smaller m are nodes in T (so that the first few levels of T are quite "full"), for rather smaller n than one might think a large fraction of windows have location whole–genome copy number 1. (For example, in *Arabidopsis*, while under 1% of length 12 windows have $WGCN_L$ 1, more than 3/4 of length 17 and 7/8 of length 22 windows do.) This implies that the deeper levels of typical T are very spindly, consisting of stretches of single–child nodes, creating long paths leading down to individual leaves whose sequence became unique at a prefix significantly shorter than typical n; most true branching is commonly over and done with in the early levels of the tree. This suggests that a form of *path compression* would be highly beneficial: when an edge leads to a node whose set of descendents that are leaves has only a

single element, that edge and all nodes and edges below it can be replaced by a (long or short) leaf link such as already described above. On traversing such an edge, S', S'', etc. are updated essentially as if the whole sequence of edges represented by the compressed edge were traversed in quick succession. A compact encoding of the now–omitted sequences of edge labels is provided by the genomic sequences themselves (which are typically rather smaller than the binary size of T and hence it is not much of an additional burden for them to be retained in RAM during mapping as well, especially since it is sufficient to retain only plus strands): the representative location for the leaf is still provided by the compressed link, and the needed edge labels are easily extracted from the tail of the window sequence referenced by it.

The efficacy of the final CokusV8A encoding of T is apparent from, e.g., statistics of the *Arabidopsis* $n = 31$ tree. There are just 213,030,377 short nodes, only 2,136 long nodes, 213,032,512 links to non–leaves, 217,121,214 compressed short links to leaves, and only 6,769,808 compressed long links to leaves. The final packed binary encoding of T occupies only 1,987,808,017 bytes (under 1.852 GiB). As this build of *Arabidopsis* has 119,707,898 bp, this is just ~16.6 bytes per basepair. Further, this is small enough for the mapping process to operate in a 32–bit address space on suitable operating systems should the need arise, and only a modest (e.g., 4 GiB) total amount of RAM is required by today's research standards. The CokusAlignment implementations included here are fully multi–threaded (with each thread working independently on a different q) and use POSIX *mmap*() to access the encoded form of T so that a single copy of T is shared among CPUs in an individual machine and interaction with components of the virtual memory system (e.g., the Unified Buffer Cache of Apple Mac OS X) of modern operating systems is efficient.

On a 32–CPU 64–bit Intel Mac OS X 10.4 cluster of three Apple Mac Pros with 8 CPUs each (dual quad–core 3.0 GHz Intel Xeon X5365, 16 GiB 667 MHz DDR2 ECC FB-DIMMs) and two Mac Pros with 4 CPUs each (dual dual–core 2.66 GHz Intel Xeon 5150, 10 GiB 667 MHz DDR2 ECC FB-DIMMs), mapping with $\alpha = \beta = 0.01$ of the 278,927,842 raw reads from 78 lanes for the *Arabidopsis* datasets presented in this work took approximately four days. This is an average of ≈200 processed q per second or ≈5 hours per lane on a single 8–CPU machine (however, see the next paragraph; due to a fair number of lanes here having suppressed cycles near the edges of reads, relatively small and simple changes to the current implementations would increase speed significantly). Time needed, of course, depends greatly on user–specified parameters such as α and β (for example, as typically every q has some non–zero — albeit often miniscule — probability of mapping to every single window, if α and β are sufficiently close to zero then the mathematics demands that all millions or billions of windows be returned for each of the hundreds of millions of q so that just the I/O required to report answers could be enormous even if there was an algorithm otherwise using zero time and space).

Currently, suppressed cycles near the start (for FW traversals) or end (for RC traversals) of a read typically slow mapping greatly due to a loss of effectiveness in the algorithmic optimization of establishing a lower bound on $M(q)$ via a quick greedy search. During greedy descent when focused on a node with the next edge to be traversed corresponding to a suppressed cycle, all edges (e.g., all four of "A", "C", "G", and "T") are numerically exactly equally attractive to the greedy search and an arbitrary (but deterministic — e.g., always "A") choice is taken, which is not unlikely to be the "wrong" choice. The result is often a poor greedy lower bound and pruning during the main queue–based traversal not being as effective as it could be, slowing mapping (but not affecting correctness). Indeed, the mapping of the 78 lanes of the previous paragraph on the five Mac Pros saw per–machine rates vary widely — between ~34 and ~4,507 reads per second across lanes — with all of the slowest lanes having suppressed cycles at the edges of reads. (In the Cokus pipeline distribution available on the web site that accompanies this

publication, sample lane `N1try2-L7` with 16 non–suppressed bases followed by a single suppressed base followed by 14 non–suppressed bases has 2,946,339 reads and maps at ~3,082 reads per second on one of the 8–CPU machines, finishing in ~16 minutes.) If the number of suppressed cycles is not too large, this issue can be addressed by having greedy search try all edges at each suppressed cycle (or, e.g., just those corresponding to non–ambiguous IUPAC nucleotides; of course, at non–suppressed cycles, a greedy choice is made as before). As the current greedy search is extremely fast, even if it were, e.g., sixteen times slower (as it might be if there are two suppressed cycles near the edges of a read), the possibility for great increase in pruning during the queue–based main search is likely to more than compensate for the additional time. As an alternative, if suppressed bases are at the very beginning or very end of reads, one may consider trimming these bases and using smaller window sizes instead.

As RAM is precious, it is also desired to represent double–ended D efficiently in memory. In the CokusV8A–based CokusAlignment implementation provided here, D is a single array that encodes two logical array–based heaps and each entry of D occupies 16 bytes as follows. An IEEE 754 single–precision floating–point value storing S' for the relevant node of T occupies 32 bits. A single–bit flag states whether the entry is part of an FW (1) or an RC (0) traversal. (Allowance for FW vs. RC is possible even in applications other than BS–Seq, although such may not be useful or appropriate; typically, in other applications every read can be taken as FW, i.e., only an FW traversal is performed for each q.) A 31–bit unsigned binary integer (0..2,147,483,647) gives a zero–based byte offset from the beginning of the packed form of T that indicates the node of T, and a 6–bit unsigned binary integer (0..63) indicates the tree depth (i.e., length) of the node. Two unsigned binary integers $i_{\text{AUX–to–MAIN}}$ and $i_{\text{MAIN–to–AUX}}$, each 29 bits (1..536,870,911), are used to cross–index the two heaps as described momentarily and enable fast priority queue operations simultaneously at both ends of D. Indexing of each heap is as usual as a binary tree stored in a 1–based array with the left and right child of index i being $2i$ and $2i + 1$, respectively (and with the priority of every heap node at least as big [or no more than] every one of its heap descendents). Entries are stored in the D array according to the high–priority–is–top ("main") heap. The low–priority–is–top ("auxiliary") heap is essentially composed of the $i_{\text{AUX–to–MAIN}}$ fields of the D entries; the remainder of the data for each entry of the auxiliary heap is in the D entry of index $i_{\text{AUX–to–MAIN}}$. To provide $O(1)$ identification of corresponding heap entries in the other direction, the auxiliary index corresponding to a main heap entry is given by the $i_{\text{MAIN–to–AUX}}$ field of that entry. For the sake of completeness (tie–breaking rules are arbitrary and do not affect correctness), the complete specification of priority used is as follows. First (as required), larger S'' is higher priority. In the case of a tie, the longer node has higher priority. If a tie remains, FW traversals are higher priority than RC. If a tie still remains, the node with smaller offset from the beginning of the packed encoding of T has higher priority (for which a tie is not possible).

Adaptations for Larger Genomes

It is now appropriate to discuss the CokusV8B representations for trees T and double–ended priority queues D that are more flexible and support larger genomes (but which are slightly less efficient for *Arabidopsis*) than CokusV8A. The 1– and 2–byte headers of short and long nodes are the same, but links are changed. The 31–bit byte offset (limited to the vicinity of 2 GiB) in links to non–leaves is increased to 39 bits to accommodate trees whose packed form is up to the vicinity of 512 GiB. Short links to leaves, formerly 32 bits but now 40 bits, handle WGCN_L in 1..8 rather than just $\text{WGCN}_\text{L} = 1$. Representative locations as 30–bit units (1–bit strand, 3–bit chromosome number limited to 0..7, 26–bit chromosomal coordinate limited to 0..67,108,863) are replaced by 35–bit units: 1–bit strand (0 = plus, 1 = minus) followed by a 34–bit unsigned binary integer (capable of the range 0..17,179,869,183) that encodes both on which genomic

DNA molecule and at what coordinate on that molecule the window of the representative location resides as described in the next paragraph. The final 3 bits of a short link communicate $WGCN_L$ in 1..8, using 000_2 for 8 and $001_2..111_2$ for 1..7. Long links to leaves are expanded from 8 bytes to 9 bytes each. The 30–bit representative location is extended to 35 bits as described for short links (and discussed further below), and this is immediately followed by a 35–bit unsigned binary integer providing for $WGCN_L$ up to 34,359,738,367 rather than the former 32–bit value limited to 4,294,967,295. In CokusV8B, all units are stored in big–endian byte order, which is essentially equivalent to a simple stream of bits.

Depending on the extent to which the reference genome from the genome project conducting sequencing is finished, there is considerable variation in the number of genomic sequences across reference genomes that one may wish to use. For example, JGI *Populus trichocarpa* 1.0 has 485,510,911 bp but spread over 22,012 scaffolds of size 1,001..35,571,569 bp, while NCBI *Homo sapiens* 34 (UCSC hg16) has 3,070,144,630 bp in only 25 molecules (chr1..22/X/Y/M) of size 16,571..246,127,941 bp. As it is relatively inefficient to reserve bitfields of large enough width for separate indexing of genomic sequences and coordinates on them to deal with a wide variety of potential usage scenarios, storage of representative locations in CokusV8B is other than just a widened version of that used for CokusV8A. Instead, a user may choose one of the many ways to form a single virtual "chromosome" (e.g., by concatenating all plus strand sequences end–to–end in some fixed but arbitrary order, with spacers if desired). Because the implementations of CokusAlignment here operate almost completely independent of the details of reference genome organization, there is, for example, no requirement that windows occur spaced every 1 base on a genomic sequence; building of *T* can be performed with windows arbitrarily and variably overlapped or separated on genomic sequences. (Of course, one of the window generators included with the present implementations of tree building needs to be altered to emit only the windows of interest. Indeed, *T* could be built from any collection of short sequences of equal length whatsoever; recall the mathematical abstraction of genomes currently in use as discussed near the beginning of this document. The restriction to equal widths is not essential, but relaxation of this constraint was not needed for the present project and hence is not elaborated here. Allowance for gapped alignments with various probability models on the gaps also theoretically fits in the CokusAlignment algorithmic and mathematical framework, but has not been explored as such has not been needed for the present project.) With a single virtual chromosome, all that is needed to express a representative location is a single–bit strand flag and a single coordinate, and the 34 bits provided for the latter allows for a range of more than 17 Gbp of virtual chromosome, which is sufficient for immediately anticipated applications. (As already mentioned, further encodings are easily implemented should even larger genomes be of interest.)

As an example relevant to the present work, the NCBI *Mus musculus* 37.1 21–chromosome assembled mouse genome (chr1..19/X/Y) has a total of 2,654,895,218 bp in molecules of 15,902,555..197,195,432 bp. The plus strand sequences were concatenated in the order chr1, …, chr19, chrX, then chrY with 100,000 bp "x" spacers between chromosomes. As the implementations of tree building included here (and described below) do not consider any window whose sequence contains at least one "x" to be part of the genome, the resultant set of windows for *T* is equivalent to that which would be obtained from the chromosomes as they originally were. (Of course, 1 bp spacers would have been sufficient, and if the implementation of tree building included here was slightly modified, inclusion of spacers for the present purpose could be dispensed with entirely.) The resulting CokusV8B tree for $n = 31$ occupies ~17.0 bytes per basepair (45,103,975,398 bytes: 3,875,407,338 short nodes, 0 long nodes [as opposed to *Arabidopsis*, the mouse sequences contain only "A", "C", "G", "T", and "N"], 3,875,407,337 links to non–leaves, 4,351,875,616 short links, and 10,239,255 long links; without path

compression, there would have been 62,636,850,826 non–leaves and 4,362,114,871 leaves). An *Arabidopsis* tree is 2,398,843,683 bytes (~2.23 GiB), just over a 20% expansion (~392 MiB) relative to CokusV8A. (This is close to 25%, an easy theoretical upper bound on the expansion of a CokusV8A tree when it is re–expressed in CokusV8B form: 1– and 2–byte node headers stay the same size, non–compressed links and short compressed links grow from 4 bytes to 5 bytes, and long compressed links go from 8 bytes to either 5 or 9 bytes; the maximum ratio among these of new to old is 5/4.)

Today, 64 GiB RAM machines are certainly available, but 16 GiB (such as that of the largest three machines mentioned above) is more common and the encoded T for large genomes, e.g., the current mouse example (at a hair over 42 GiB), exceed this smaller amount. One could rely on the virtual memory subsystem of modern operating systems to manage this issue via demand paging. This is trivial to implement, as essentially nothing need be done other than compilation for and use of a 64–bit platform. However, in typical applications, to have a chance to avoid the enormous performance penalty associated with such an approach, careful arrangement of the order of nodes in the encoded form of T, the order of presented q, and a degree of luck would be required to attain sufficient spatiotemporal locality of reference; heavy random access to modern hard drives is orders of magnitude slower than such accesses to modern DIMMs. (Of course, if Moore's Law — now in its fifth decade — continues unabated, machines with 64 GiB and more of RAM will soon become commonplace. If the trend sustains itself sufficiently long, brute force approaches to mapping may become economically inexpensive and perhaps even preferred due to their simplicity. However, that day is not yet today, and some anticipated extensions increase the number of possible alignments to such an extent that feasibility of brute force is expected to be only in the remote future at best.)

Other strategies readily present themselves, however. For example, S' and S'' do not change on those nodes that remain under reduction to a subset of windows, and so another possibility is to partition windows so as to form smaller virtual genomes whose encodings of individual T are small enough to completely reside in available RAM. One may combine the outputs of multiple CokusAlignment runs (see the next two paragraphs), one run for each virtual genome component of the original genome, to obtain mathematically equal information on L and M as would be had without partitioning. (For example, note that $M(q)$ for the original genome is equal to the maximum $M(q)$ over the individual component runs.) The main disadvantage of partitioning approaches is that pruning is generally not as effective (and, hence, mapping is slower) because the high S' and S'' for a given q typically often lie only in a single partition component. Various strategies to combat this are easily imagined, e.g., the establishment of greedy lower bounds can be broken out into a separate initial phase: one genome component would be resident in a given machine's RAM at a time, and lower bounds computed for all q. Once all components are processed, lower bounds mathematically equal to those without partitioning are easily obtained by taking for each q the maximum lower bound obtained over all components. These cross–component lower bounds would be made available to all components during a second phase in which execution of the main, rigorous mapping loop would be made for a single component at a time per machine. (However, for the large genome applications encountered thus far, pure partitioning with no attempt at recovering lost pruning power has been found to be adequate.)

Some terminology to be introduced presently simplifies discussion in the next paragraph. Suppose ("α–mapping") that one wishes to obtain for each read all hits scoring at least α. (Both α and β will generally be assumed to be members of (0, 1) or [0, 1] here.) This may be done by ("strategy 1") disabling β pruning, or by ("strategy 2") setting β to the same value as α and filtering out any and all extra hits (i.e., those scoring below α), but strategy 2 is generally

less efficient than strategy 1. Suppose instead ("α–top–hit–mapping") that one wishes to obtain for each read a single, top hit or determine that the score of top hits is below α. This can be achieved by ("strategy 3") disabling β pruning and stopping traversal at the first leaf visited (if any), or by ("strategy 4") the typically slightly less efficient method of setting $\beta = 1$ and filtering out any and all extra hits (i.e., all hits except the first for each read with at least one hit). Usual mapping is referred to as "(α, β)–mapping" if needed.

Combining the outputs of multiple CokusAlignment runs to obtain results mathematically equal to a single unpartitioned run may be performed as follows, where α' and β' are the original values of α and β (which are generally both assumed to be in (0, 1) here), and $M'(q)$ for read q refers to the value of $M(q)$ in the unpartitioned genome. A simple approach ("method 1") is to (β' α')–map once with each genome component, concatenate all results (noting that for every read q one can now easily calculate $M'(q)$ or determine that it is below α'), and finally filter out any extra hits (i.e., those below $\beta' M'(q)$ but at or above the generally smaller $\beta' \alpha'$) for those reads q with $M'(q) \geq \alpha'$. Method 1 typically often assumes a worst–case value of α' for each $M'(q)$ and so not as much pruning is made as is generally possible. A conceivably more efficient alternative ("method 2") is to perform two passes, the first being α'–top–hit–mapping once with each genome component, so that resolution of whether $M'(q) \geq \alpha'$ or not and a numerical value for $M'(q)$ in the former cases becomes easily available for every read q. Then, in a second pass, α–mapping with α taken per read to be $\beta' M'(q)$ is performed once per genome component and all results concatenated. For the present work, method 1 was used in combination with strategy 2 and, as described thus far, this would essentially reduce to a single ($\beta' \alpha'$, $\beta' \alpha'$)–mapping in each genomic component. However, β in these mappings may be increased from $\beta' \alpha'$ to β' (generally resulting in a performance improvement) without loss: for reads q with $M(q) \geq \alpha'$ in the current genome component, hits with scores at least $\beta' M(q)$ are sufficient (since $M'(q) \geq M(q)$); for reads q with $M(q)$ in [$\beta' \alpha'$, α'), hits with scores at least $\beta' \alpha'$ are sufficient (as either $M'(q) \geq \alpha'$ or q will have no hits retained in the final combined output; hence, hits with scores at least $\beta' M(q)$ [which is smaller than $\beta' \alpha'$] are also sufficient); and no hits are needed from reads q with $M(q)$ below $\beta' \alpha'$. As 0.01 was the desired value of both α' and β', (0.0001, 0.01)–mappings were performed on each mouse genomic component below.

For $n = 31$, windows of the mouse genome were partitioned into five components based on first letter: "A", "C", "G", "T", and "other". (While not giving a particularly equal division — the "other" group is very small and the C + G content of mouse is ~42% so that the "A" and "T" components are significantly larger than the "C" and "G" components — as long as the largest component is reasonable for available RAM, this is not a critical concern. A better division would be to take sets of windows from four contiguous [and slightly overlapping] covering regions approximately equal in length from the concatenated single mouse virtual chromosome. This would not only produce trees of more uniform size, but would also allow additional memory to be saved as only one of these genomic regions would need to be in memory at a time rather than the entire original virtual chromosome.) The resultant CokusV8B trees are ~12.27 GiB, ~8.75 GiB, ~8.76 GiB, ~12.23 GiB, and ~562 KiB, respectively, which are quite usable on 8–way 16 GiB machines (even with ~2.47 GiB of the somewhat–inefficiently–encoded 1–byte–per–basepair plus strands of mouse genome and live portions of 8 threads' double–ended D also in RAM). To enhance performance, it has been found helpful to POSIX *mlock*() as much of the genomic sequences and T as possible, as this communicates to the operating system the great importance of keeping these in physical RAM even though they may be very large and consume almost all of physical RAM. (This also has the advantage of ramping up to full mapping speed faster when processing many q since *mlock*() typically reads underlying memory mapped files sequentially if they are not already in cache and few page

faults and userland vs. kernel context switches tend to be incurred; otherwise, the many VM pages needed are typically first hit in a highly scattered order that is slow to load from disk and has relatively high overhead. The main disadvantage is that if only a handful of *q* are to be mapped, then all locked data must be read even if it will not be accessed.)

The format of double–ended *D* entries must also be modified to accommodate larger genomes; the 31–bit byte offset into *T* is expanded to 39 bits (0..549,755,813,887). Finally, a smaller detail (in both implementations) not yet mentioned concerns the actual number of simultaneously live elements over the lifetime of each per–thread *D* being hard to anticipate (and theoretically very large). To avoid having to move elements should an expansion be required, virtual memory for a maximum–sized *D* is allocated in each thread as soon as execution begins. This costs very little, as allocation of even many tens of gigabytes of virtual memory address space is extremely cheap on 64–bit platforms when individual memory pages are only truly allocated on demand as referenced.

Tree Construction Overview

Although the rationale, ideas, usage, and encoding of trees *T* have now been discussed, it remains to describe how typical multi–gigabyte information–dense tree files with nodes of variable length are constructed. Tree construction is a non–trivial process (and obviously essential for the method as CokusAlignment depends on its existence) for largely the same reasons as why traditional (e.g., pointer–based) encodings are impractical as already discussed above (i.e., it is difficult to work internally in–core: machines having the colossal amounts of RAM required are not readily available at this time).

On the other hand, modern commodity SATA hard disks are extremely cheap immense mass storage devices at approximately 200 USD per terabyte. (For comparison, it is helpful to keep in mind that individual Solexa runs of eight lanes currently stand at thousands of dollars each.) While placing, e.g., four spindles into service as a RAID 0 stripe set may give under 10 MB/sec during heavy random access, upwards of ≈200 MB/sec can be available for largely sequential "streaming" access — one might view the disks of today as the modern counterparts of the reel–to–reel magnetic tapes of yesterday. This latter figure is, of course, still nowhere near comparable to modern RAM subsystems which achieve gigabytes per second and with virtually no random access penalties, but is still in excess of 17 TB per day.

It follows that external algorithms that operate by making several sequential passes over a handful of different streams would leverage the three advantages of present–day external storage: terabytes of capacity, moderately fast sequential access, and low price. A relevant and excellent example is provided by the computer science problem that is perhaps the one that has been the most extensively studied: sorting. It is routine to use, e.g., GNU `sort` on modern hardware to sort the lines of individual plain text files of even hundreds of gigabytes by configurable criteria in a matter of minutes or hours (but, of course, having a few gigabytes of internal memory to serve as in–core sorting buffer is not unimportant). Indeed, the ability to sort very large files is used in the first stage of tree construction as well as peripheral tasks to be outlined later. Subsequent stages of tree construction are fully streaming and need to keep only a small constant number (e.g., 1 or 2) of input lines in–core at a time.

Main Ideas of Tree Construction

In the first stage, a two–column plain text stream enumerating all windows with the first column being window sequence and second column being genomic location is generated and

piped to GNU `sort` to be sorted lexicographically by window sequence (with ties broken by location). The result, file *U*, is a representation of W_n with lines corresponding to each element of G_n appearing consecutively, and this file is permanently saved on disk as it is useful for more than just tree construction (see *Solutions of Peripheral Computing Problems* below). The typically large file size — e.g., 10,294,861,168 bytes in 239,415,376 lines for *Arabidopsis* $n = 31$ and 233,630,723,744 bytes in 5,309,789,176 lines for mouse $n = 31$ — can be mitigated (at a typical cost of up to several CPU hours, although if desired this can be concurrent with subsequent stages) by, e.g., GNU `gzip --best` (reducing *Arabidopsis* to under 2.92 GiB and mouse to under 59.8 GiB; while other compression programs can achieve better compression, `gzip` represents a good compromise between compression time and performance and, further, decompression with `gzcat` is relatively fast and it streams its I/O well for subsequent direct use of compressed files in UNIX pipelines). Sorting generally requires temporary disk space comparable to input size. Current implementations of window generation bring whole chromosomal sequences into core; this could be reduced, but to little gain as mapping will have plus strands, the finished form of *T*, and multiple *D* in RAM, which together are typically much larger.

The second stage begins by reading the output of the first to form file E_n. An iterative process is begun by which file E_{i-1} is produced from E_i and file Y_{i-1} is produced from E_{i-1} and E_i for successive $i = n, n-1, \ldots, 1$. The *E*–files are large (the largest for *Arabidopsis* is ~17.5 GiB) and deleted as soon as possible (e.g., E_n is deleted as soon as Y_{n-1} is generated). Y_i for i in $0..(n-1)$ is the binary encoded nodes of *T* of length i, except that when Y_i is generated, the final byte lengths of it and more shallow layers of *T* are not yet known, and so byte offsets to non–leaves (these being all to nodes in the next deeper layer) are all given as relative to the beginning of the next deeper layer rather than the beginning of *T*. In the third stage, the final byte lengths of all layers of *T* are known, and so the *Y*–files can be concatenated (first Y_0, then Y_1, etc.) and appropriate adjustments applied to byte offsets as node links stream by to create the final, finished encoded form of *T*. (This has a net effect to order the nodes of *T* according to a breadth–first search [BFS]; the performance impact on mapping of other orders, e.g., that of a depth–first search [DFS], has not yet been investigated. BFS orders have conceivable advantages: for example, if *T* is barely too large for available RAM, then because the upper layers of *T* consisting of those nodes with shortest length are typically frequently referenced [as the first steps of processing a typical *q* must touch them] and a BFS order places these layers into an interval of memory addresses, they can be easily and profitably locked with *mlock*(). DFS orders, however, attempt to provide address locality for the subtree of descendents of each node, which might be helpful as regards demand paging and the memory cache hierarchy for the near–final steps of processing typical *q*.)

The format of the *E*–files is plain text lines of five columns each: node (that is, element of G_n^* that may or may not appear in the final encoded form of *T* due to path compression) given by an explicit nucleotide sequence; a representative location for this node; the $\mathrm{WGCN_L}$ for this node (i.e., the number of elements of W_n whose sequence is an extension of the first column); a 15–character string (the *extension flags*) to be explained momentarily and in the next paragraph that is either "`ACGTVHDBMRYKWSN`" with zero or more letters in lowercase and zero or more letters each replaced with an ASCII hyphen "`-`", or is all "`x`"s; and the *sequence whole–genome copy number* $\mathrm{WGCN_S}$ of this node, respectively. This last column is defined to be the number of elements of G_n whose sequence is an extension of the first column. Due to the sorted nature of the output from the first stage, creation of E_n — essentially corresponding to the passage from W_n to G_n — is trivial: in a single pass over the output from the first stage, the lines that share each window sequence appear consecutively, and thus are easy to identify as groups. The window location of the first line in each group is taken as representative location for the group.

Each successive group generates the next line in E_n with the first column being the common sequence, the second column being the representative location for the group, the third column being the number of lines in the group, the extension flags all "x"s (as there are no window sequences that strictly extend the current sequence, which is of full length n), and the last column obviously being 1.

The streaming generation of E_{i-1} from E_i is as follows. Due to the lexicographic order used in the first stage, lines that share leading $i - 1$ characters of sequence appear consecutively, so that, in a similar manner to that already described above, such subsets of lines are easily identified as groups during streaming. Each successive input group generates the next line of E_{i-1} as follows: the new first column is the common sequence prefix of length $i - 1$, the new representative location is taken to be that of the first line in the group, the new $\mathrm{WGCN_L}$ is the sum of the same over the lines in the group, and the new $\mathrm{WGCN_S}$ is the sum of the same over the lines in the group. The new extension flags point the way to the sequences of the input group: a letter is present (i.e., is a non–hyphen) if and only if a member (and there is at most one) of the input group has its sequence end in that letter. Further, there is enough information immediately available to determine, should the new entry of E_{i-1} be a node appearing in the final encoding of T, whether each letter in the new extension flags will be a leaf–link or not and, if so, if that link will be long or short. Critical advantage is taken of this, and extension flag letters that correspond to what would be long links, i.e., those whose corresponding input line have $\mathrm{WGCN_S} = 1$ and $\mathrm{WGCN_L}$ too large (greater than 1 for CokusV8A, greater than 8 for CokusV8B) are emitted in uppercase, with the rest in lowercase.

The production of near–final encoded tree nodes and links in the form of the Y–files is relatively easy due to the careful arrangements made above. Recall that to stream–produce Y_i for given i in $0..(n-1)$, assistance in the form of streamable inputs E_i and E_{i+1} is available. Proceeding through both input streams in synchrony, each E_i line pairs with a readily–identifiable group of E_{i+1} lines. As only non–leaf nodes not path–compressed–away appear explicitly in the encoded form of T, pairs whose E_i line has $\mathrm{WGCN_S} = 1$ are skipped. A non–skipped pair corresponds to a non–leaf node not path–compressed–away, and the 1– or 2–byte header (indicating short or long node and the subset of potential links present) is trivially generated from the extension flags of its E_i line. In the canonical order described earlier, the links are then emitted as follows. First, the number of bytes z (if any) used by the target of the link in the encoded form of the next deeper layer (if any) of T is determined: if the $\mathrm{WGCN_S}$ of the E_{i+1} line is 1, then $z = 0$ as that line does not appear explicitly; otherwise, that line appears explicitly as a node and from its extension flags one can tell whether it is a short or long node and how many long vs. non–leaf/short links it has, which collectively determine z. If $z = 0$, one emits a short or long link to a leaf into Y_i (as the E_i line is a node that will appear in the final form of T but the child E_{i+1} line will not); the contents of the E_{i+1} line are enough to determine short vs. long and generate the link. Finally, if $z \neq 0$, a link to a non–leaf not path–compressed–away is emitted into Y_i (as both the E_i and E_{i+1} lines are nodes appearing in the final form of T), except that the best that is easily done for the byte offset is relative to the start of Y_{i+1} by keeping a running sum of all z computed in this iteration thus far; fix–up is performed in the third and final stage as already described above.

Future versions of tree construction may benefit from incorporation of ideas from existing strategies for fast generation of suffix trees. However, the above method is relatively simple to understand and implement, and has proven readily usable for current applications and is expected to be sufficient for the near future.

Miscellany and Solutions of Peripheral Computing Problems

IUPAC nucleotides are represented internally with "A", "C", "G", "T", "V", "H", "D", "B", "M", "R", "Y", "K", "W", "S", "N", and "x" (this last letter being added to represent the empty set, i.e., a non–nucleotide such as a spacer) being unsigned binary $0000_2..1111_2$, respectively. This code has a number of convenient properties. The nucleotide complementary to h in 0..15 is h XOR 011_2 if $h < 12$ and h otherwise. The unambiguous nucleotides are the two bit quantities $00_2..11_2$, and letters of equal cardinality (e.g., "V" corresponds to the subset {"A", "C", "G"} of cardinality 3) are intervals: 15..15 has cardinality 0, 0..3 have cardinality 1, 8..13 have cardinality 2, 4..7 have cardinality 3, and 14..14 has cardinality 4. To avoid control and whitespace characters externally, 0..15 are biased by +33 to become successive characters of the 16–character ASCII string "`!"#$%&'()*+,-./0`". If needed, answering questions such as "is 'A' permitted by this (potentially ambiguous) nucleotide?" can often be reduced to simple "bit–fiddling" (which can be preferred to use of lookup table arrays on today's CPUs, which typically have a surplus of ALU resources and a deficit of access to caches/memory, often making direct computation more desirable than indirect memory references). Using ISO/IEC C99, for example, "A" is permitted in h if and only if `(0x5371U >> h) & 0x1U`; replace `0x5371U` with `0x65B2U` for "C", `0x6AD4U` for "G", and `0x5CE8U` for "T". A bitfield with simultaneous answers for "A" (msb) then "C" then "G" then "T" (lsb) could be obtained via `(0x0F6935AC7BDE1248ULL >> (h << 2)) & 0xFU`.

UNIX pipelines and POSIX FIFOs and `tee` are used in concurrent processes to reduce the number of times *E*–files need to be read from disk. (This saves time, but uses more temporary disk space.) One of the Mac Pro machines already mentioned has four contemporary 500 GB hard disks in RAID 0 with a 64 KiB stripe size; on this machine, total tree building time for all three stages for $n = 31$ is under three hours for *Arabidopsis* (with only ~16 minutes in the first stage) and about two days for mouse. It would be relatively easy to reduce these times by a factor of two to four; however, as tree building was needed only infrequently for the present project, current tree building times were not prohibitive and optimization effort was expended elsewhere (namely, in the main implementations of CokusAlignment). The easiest optimizations would be to reduce the amount of disk I/O required, such as by switching from the convenient–but–somewhat–inefficient plain text encoding of intermediate *E*–files used presently to either a fixed–width binary encoding (e.g., four bits per nucleotide letter and packed binary for locations, extension flags, and integers) or a variable–width compressed binary encoding (e.g., only two bits for the most common nucleotide letters and short forms of common values of other fields).

The current implementation of tree building for larger genomes uses 10 decimal digits for virtual chromosome coordinates. This saves a slight amount of temporary disk space during tree construction for genomes less than ~10 Gbp at the expense of not covering the entire ~17.2 Gbp range supported by the CokusV8B tree format, but is easy to change if needed.

Evidence of the compactness of the final encodings of *T* is provided by the relatively low compressibility of the binary files for *T*. For $n = 31$ for *Arabidopsis*, only ~14.2% is saved by `bzip2 --best` (which uses Burrows–Wheeler block sorting and Huffman coding), ~24.2% by `gzip --best` (which uses Lempel–Ziv 1977 and Huffman coding, unusually beating `bzip2`), and ~44.7% by the world–class but ultra–slow context mixing `paq8o8 -8` (taking more than four CPU days to compress compared to about seven CPU minutes for the others). However, while the lack of rapid random access to decompressed data (and, in some cases, substantial consumption of RAM — even more than 1 GB — for the decompression process itself) precludes direct use of these compression schemes to conserve additional RAM during mapping, that these percentages are positive suggests that there may still be some simple patterns in *T*

that remain to be exploited without appreciable decrease in mapping performance for T that already fit in core. (Of course, a very compact "encoding" of T is decompressor source code or executable together with a compressed version of the plus strands of the reference genome and source or executables for the tree building process; "decompression" then involves expansion by the outer compressor followed by the entire tree building process as already described. Using `paq8o8 -8` for the outer compressor achieves a size under 25.8 MiB — less than 1.4% of the binary tree file — for $n = 31$ for *Arabidopsis*. However, this obviously blatantly fails the mapping requirement of rapid random access to nodes and edges of T.)

There are a number of peripheral, less challenging computational tasks not yet discussed for which sample implementations are not provided. These are mostly reduced to the sorting of large files and various single–pass streaming procedures in the same spirit as tree building. For example, each line of CokusAlignment output in current implementations concerns one member of G_n and representative locations are given only when the $\mathrm{WGCN_L}$ for this member is 1 (as then the representative location is the only location); otherwise, the only location information reported is the $\mathrm{WGCN_L}$ as a positive integer. If desired, expansion to a complete list of explicit locations for lines with $\mathrm{WGCN_L} \geq 2$ is easily performed in a streaming fashion after mapping as follows. CokusAlignment output is sorted according to window sequence in the same order used for the output U of the first stage of tree construction. Then, in a single synchronized pass through the prepared CokusAlignment output and U, expansion from G_n back to W_n begins by collecting lines of U into groups according to common window sequence (which is not difficult as the order of lines in U ensures groups are composed of consecutive lines). U groups are skipped as long as their window sequence comes before that of the current mapping line in the sorting order. Otherwise, the window sequence of the current mapping line and that of the U group must be the same, and the locations given in the second column of the current U group lines are exactly those desired. Each line of mapping output includes two positive integer fields, one giving the line number the corresponding read had in the mapping input and the other a per–read serial number for mapping output lines (as, in general, a single read to be mapped produces multiple mapping output lines). Assuming these fields are propagated during location expansion, a numeric lexicographic sort on them easily enables permutation of the lines of the expanded mapping output into an order consistent with the order of presentation of q during mapping and the per–read order of visitations to leaves. (A final sort similar to this is generally required even if location expansion is not performed, as current implementations of CokusAlignment are permitted to mingle output lines from different threads in unspecified order in order to reduce output buffering and cross–CPU synchronization.)

If needed, a reasonably efficient device for having G_n to W_n expansion information available during mapping is as follows. (Since the RAM cost of the following is substantial and expansion in the applications of the present work is easily performed after mapping in a streaming fashion as already described in the previous paragraph, the following was not used in the present work.) Starting with the CokusV8B encoding of T, the distinction between short and long links is removed; instead, a link to a leaf is a 40–bit unit consisting of a 1 bit (to indicate link to a leaf) followed by a 4–bit unsigned binary integer c (in 0..15) followed by a 35–bit unsigned binary integer p (in 0..34,359,738,367). Let $m \geq 1$ be the location whole–genome copy number of the target leaf. If m is in 1..15, then $c = m$; otherwise, $c = 0$. When $m = 1$, the representative location (in this case, the only location in the whole genome for its sequence) is encoded into 35–bit p as usual. Otherwise, p contains a zero–based index into a run of m consecutive elements of a new auxiliary array A of 40–bit units. Every element of A belongs to exactly one run; the order of runs is unspecified (although when proceeding from the beginning of the binary encoding of T to the end it would be reasonable for runs to occur in the same order as do the links that reference them). The lower 35 bits of each element of A encode a window location in the usual

manner as for representative locations and, unless otherwise specified, the upper five bits of each element of *A* are all zeros. The locations in a run precisely enumerate the genomic windows that share the sequence of the leaf (there being *m* such windows); the order of windows within each run could be unspecified, but it is nice to use the same order as that used in tree construction (which, for example, places the representative location first). When $m \geq 16$, then *m* is treated as a 35–bit unsigned binary integer, partitioned from msb to lsb into a seven–element sequence of fragments that are five consecutive bits each, and the fragments placed in order into the upper five bits of the first seven entries of *A* in the run. Finally, if one is concerned over reduction of memory locality from splitting the data of any given link across two possible data structures, *A* can be folded into *T* at a cost of additional complexity (especially to tree building).

The net effect of the previous paragraph is to enable $O(1)$ access during mapping to the $\mathrm{WGCN_L}$ and representative location of any given leaf as well as best–possible $O(m)$ access to the *m* genomic locations that share its sequence at a net cost (compared to pure CokusV8B) of five bytes per genomic window with $\mathrm{WGCN_L} \geq 2$ minus four bytes per former long link. For $n = 31$ CokusV8B unpartitioned ("whole") mouse, 947,674,305 windows of 5,309,789,176 do not have a unique sequence and there are 10,239,255 long links; hence, an additional ~4.37 GiB would be required. This RAM cost could be mitigated by more efficient encodings of genomic sequences. In current implementations, each plus strand IUPAC genomic nucleotide (coded as a 4–bit quantity in 0..15 as already explained) occupies an entire 8–bit byte, while two nucleotides could be easily packed into each byte. This would save, e.g., more than 1.2 GiB for whole mouse, and further improvements are discussed in the next paragraph.

A large majority of nucleotides are typically unambiguous (i.e., "A", "C", "G", or "T", coded as 2–bit quantities 0..3). For whole mouse, fully ~96% of nucleotides are unambiguous. Hence, more complex schemes should be able to pack nearly four nucleotides per byte, saving ≈1.8 GiB for whole mouse (but rapid random access must be maintained). A relatively easy to implement approach that is effective when the fraction of ambiguous nucleotides is small and ambiguous nucleotides form few distinct window sequences is to partition (or further partition) genomic windows into those with at least one ambiguous nucleotide and those with zero ambiguous nucleotides. As ambiguous nucleotides are never referenced while mapping the latter, they can be arbitrarily replaced with unambiguous nucleotides (e.g., all "A"s) and genomic nucleotides then easily encoded at a uniform ratio of four per byte. For mapping the former, a set of distinct window sequences can be concatenated end–to–end (without expending any effort to overlap them even if savings by doing so is possible) to form a virtual sequence that is typically very small: for whole mouse, such a set has just 57,369 sequences (192,737,508 — or ~99.96% — of the 192,807,870 windows that have at least one ambiguous nucleotide consist of all "N"s) and so only ~1.7 MiB is needed. (If desired, virtual positions past the end may be used as stand–ins for locations of repeated ambiguous window sequences beyond the representative locations.) The partitions (or additional partitions) introduced here need not incur the full complexity of combining multiple mapping outputs into a single output mathematically equivalent to not partitioning (as already discussed earlier in this document) as mapping implementations may be modified to load multiple trees and virtual chromosomes into memory at the same time, with elements of *D* extended to enable simultaneous traversal in multiple trees (so as to collectively treat them as a single logical tree).

Another peripheral task is the determination of which aligned windows are PBC with at least one other window (so that, e.g., the *q* that give rise to such hits may be discarded as not mapping BS–uniquely). The *BS location whole–genome copy number* $\mathrm{WGCN_{L–BS}}$ of *g* in G_n is the number of windows *w* in W_n such that *g* shares at least one PBP with the window sequence of *w*. (One might also define the *BS sequence whole–genome copy number* $\mathrm{WGCN_{S–BS}}$ of *g*

in G_n to be the number of g' in G_n such that g shares at least one PBP with g'.) If g contains $m_1 \geq 0$ and the window sequence of w contains $m_2 \geq 0$ "C"s, then one does not need to compare all $2^{m_1}2^{m_2}$ pairs of PBPs to determine if at least one is shared. (If g and the sequence of w are poly–"C", this would be 4^n pairs to be compared simply to determine PBC of a single g and a single w, and there can be hundreds of millions of g and hundreds of millions or billions of w. For $n = 31$, $4^n > 10^{18}$ so that BS–unique filtering could involve an infeasible $10^{\sim 36}$ comparisons.) Instead, note that g and the window sequence of w share at least one PBP if and only if the two sequences are literally identical after each and every "C" (if any) in g or the window sequence of w is replaced with "T". This characterization renders sharing of at least one PBP decidable in time proportional to n, and by sorting on the first column of a two–column plain text file similar to that used as input to the first tree construction stage (with lines enumerating all windows, the first column being sequences and the second column being genomic locations) except with every "C" in the first column replaced with "T", the $\mathrm{WGCN}_{\mathrm{L-BS}}$ of every member of W_n can be computed in just the time required for a single pass through the output of the sort: identical first–column sequences have been collected by the sort into runs of consecutive lines, so that the problem is solved by counting the number of lines in each run (this being the common $\mathrm{WGCN}_{\mathrm{L-BS}}$ of every location in the run) and emitting this count paired with each of the locations in the second column of the lines of the run. If desired, a second sort can be performed on genomic locations, so that the $\mathrm{WGCN}_{\mathrm{L-BS}}$ of every w in W_n is directly available by chromosomal position. Windows with $\mathrm{WGCN}_{\mathrm{L-BS}}$ greater than 1 are exactly those that are not BS–unique, and external sorting has again been shown as an efficient way to effectively perform relational database operations (e.g., joins) on tables consisting of large sets of rows; ordering rows appropriately often renders such operations trivial (and original row order can always be restored if desired by a final sort, having introduced a serial number column or the equivalent at the outset if needed).

Concluding Remarks on Alignment

The theory and implementations provided here are suitable or adaptable for processing data resulting from the bisulfite library strategy of the present work, other bisulfite library strategies, general genomic re–sequencing (e.g., SNP detection), gene expression (either via general mRNA extraction or reduced collections of tags), chromatin immunoprecipitation, and other Solexa–based and non–Solexa experimental designs. For some applications, it may be desirable to modify the representation of T (e.g., imbuing leaves or general nodes with additional or alternative information, or extending support to even larger genomes or meta–genomes) or the traversal of T so that information about L and M not necessarily that exposited in detail for the current work is computed efficiently. The present document is hoped to elucidate both the underlying ideas and computational details of the provided implementations sufficiently and indicate some of the many anticipated extensions, enhancements, and alternatives so that such adjustments are relatively easy; the present codes are essentially research prototypes on which the author expects to continue development. As computing machines gain increasing amounts of needed random–access memory, more usage scenarios will be able to be simultaneously supported, reducing the need for separate representations of T and alignment implementations. As code matures, ease of use for end–users will eventually be addressed. At this early stage when theory races ahead of practice, prioritization restricts development to immediate needs. To conclude, a few examples of adjustments are given below.

A simple example of a trade–off that was made in the current implementations is in the structure of elements of double–ended D. As all nodes in T at a given tree depth (i.e., all nodes of a given prefix length) appear consecutively in the binary packed form of T (due to the current use of a BFS order), it might be reasonable to forego the presence of the 6–bit field (0..63) in

each element of D that explicitly gives the tree depth of the node associated with the entry. If omitted, recovery of the field would be possible, e.g., by a variant of binary search to identify the depth byte interval that the node's byte offset inhabits. This would be (probably only slightly) slower and hence trade time for a small space savings (but liberation of 6 bits is not enough to shrink elements by a much more convenient 8–bit byte unit; code complexity would necessarily increase somewhat) and practically unlimited depth range (although this is much more easily achieved by simply enlarging D elements by another byte and distributing the now–surplus 8 bits among the fields so as to also prepare D for even larger genomes). In general, the current binary formats of elements of D and nodes of T are felt to represent carefully balanced compromises.

For SNP detection, one may want to give up to a small number k (e.g., 0, 1, 2, 3, …) of "mismatches" for "free," i.e., to have up to k bases effectively dynamically suppressed from scoring chosen independently for each q so as to maximize the score of q. This can be done by extending each element of D to have $k + 1$ in–progress scores instead of 1. The jth score with j in $0..k$ would be the best score–in–progress possible when suppressing exactly j bases for the current prefix. Elements would be prioritized based on the maximum of these scores, and when traversing an edge of T, it would effectively be done both ways (either using or not using another suppressed base if any remain available); the $k + 1$ scores for each child are easily computed from the $k + 1$ scores for the parent. This is effectively a dynamic programming algorithm, and like any such, if more than just the final optimal value is desired, either sufficient backtracking information can be encoded into each element of D to allow recovery of the optimal subset of suppressed bases upon reaching a leaf (e.g., by also keeping an n–sized bitset for each of the $k + 1$ scores indicating a maximizing set of live bases), or such can be re–computed on demand as needed.

Solexa's ELAND mapping (ranking hits based on the number of discrete mismatches between discrete genomic and discrete read basecalls) can be subsumed with the use of simpler scoring factors in S' and S''. It is easy to adapt the CokusAlignment implementations here so that the entries of a 16–by–16 matrix $V_{i,j}$ of reals in [0, 1] are the scoring factors, with $V_{i,j}$ used when the discrete read base is i in 0..15 and the discrete genomic base is j in 0..15 (using the encoding of IUPAC nucleotides discussed above). Taking $V_{i,j} = 1$ when at least one of "A", "C", "G", or "T" is consistent with both i and j and $V_{i,j} = \mu$ in [0, 1] (e.g., 1/2) otherwise, then $S'(x)$ for a leaf x of T is μ^k in [0, 1], where k is the number of discrete mismatches between the read sequence and x. Dropping pruning based on β and keeping just that of α, use of an appropriate absolute scoring threshold for α translates into finding all genomic matches that disagree with the read ("have errors") in up to a user–configurable number of bases. (Note that care must be taken in the choice of μ so that μ^k for k of interest are distinct in the floating–point arithmetic used, but this is not a problem in typical applications; for example, $(1/2)^{0..64}$ are all distinct [and even exactly representable] in IEEE 754 single precision. If needed, passage from multiplication of linear scores to addition of log–scores could be made in the implementation to better handle alternate scoring systems such as these, or entries of D could be modified to directly store, e.g., k. Logarithms also help avoid denormal floating–point numbers and underflows (which can be slow on today's platforms) if one does not wish to flush to zero and multiplications are chained to the extent that many values extremely close to zero are produced. Finally, note that due to potential floating–point representation and rounding errors, if two consecutive discrete final scores are x and y with $x < y$ and one wishes y to be an inclusive lower mathematical threshold, it is generally safer to use, e.g., $(x+y)/2$ as a computational threshold.) If β pruning is retained, then, e.g., setting $\beta = 1$ will find all hits that have at most as many errors as allowed by α but only give those with the fewest number of errors. Other appropriate choices of β permit finding for each read all hits that have at most a specified number of errors in excess

of the minimum number (should the minimum be permitted by α). For V, one could also use substitution matrices similar to those of BLAST or informed by nucleotide substitution rates.

One may view the result of a chromatin immunoprecipitation experiment as determining a density (a non–negative real number ideally proportional to true counts of library fragments) on genomic sequences. This density is "smeared" both due to fundamental mapping ambiguities (i.e., window sequences with $\mathrm{WGCN_L} > 1$) as well as ambiguities of location within fragments as only fragment ends are sequenced and the ends may not have caused or inhibited precipitation. (However, the probability distribution of fragment lengths can be determined from analysis of a gel electrophoresis image of the library, and hence the fragment length distribution can be assumed to be known from gel densitrometry.) Once mapping produces a preliminary density, fragment length ambiguity is readily accounted for via a certain convolution of the mapped density and a particular probability distribution easily derived from the empirical fragment length distribution. While it is possible to construct a mapped density with α and β parameters as used here for BS–Seq, one may instead want for each q a sufficiently large set of w in W_n such that the sum of $L_q(w)$ over these w is at least some absolute level δ, such as 0.99 (i.e., that almost all of the posterior distribution on location for each read is captured so that computed density is rigorously close to that which would be obtained by full brute force calculation).

Such a desire can be accommodated with a new pruning strategy. Denote by m the total number of windows and note that CokusAlignment visits the leaves of T also in order of descending L_q. Each visit to a leaf x affords $S'(x)$ and so contributes information about both the numerator of $L_q(w)$ for windows whose sequence is x as well as the denominator of all L_q. The problem essentially mathematically reduces to having elements of a finite sequence $(z_1, \ldots, z_m)$ of m non–increasing non–negative reals with $z_1 > 0$ available successively and deciding when enough elements k in $1..m$ have been seen so that their sum is at least a pre–specified fraction δ in (0, 1) of the unknown total $r_{\mathrm{TRUE}} := z_1 + \ldots + z_m > 0$ of all the elements. ($\mathrm{WGCN_L} > 1$ can be understood as effectively repeating $S'(x)$ for a given x that many times.) As the elements are non–increasing, stopping at z_k guarantees all of the following elements are each individually at most z_k, and so with $r_{\mathrm{SO–FAR}} := z_1 + \ldots + z_k > 0$, one has r_{TRUE} in the closed real interval $r_{\mathrm{SO–FAR}} + [0, (m–k)z_k]$. As z_k typically falls rapidly, this interval is often soon narrow so that the uncertainty in the denominators of L_q quickly becomes proportionally small. Indeed, deciding if $(m–k)z_k / r_{\mathrm{SO–FAR}} \leq (1–\delta)/\delta$ does not depend on knowing r_{TRUE} yet implies $r_{\mathrm{SO–FAR}}$ is at least δ fraction of r_{TRUE}. Further, numerators of $L_q(w)$ for w in W_n whose window sequence is one of those visited are known exactly while their denominators are in $[1, 1/\delta]\, r_{\mathrm{SO–FAR}}$, so that if $r_{\mathrm{SO–FAR}}$ is used for the denominator then the computed $L_q(w)$ values are in $[\delta, 1]$ times full brute force values so that their accuracy is controllably high. In addition, the sum of $L_q(w)$ over all windows whose sequence is not visited is at most $1 – \delta$ and so is controllably small.

Hence, global control over missing computed mapped density relative to theoretically perfectly–computed mapped density is attained (as at most $1 – \delta$ fraction of all density is missing since this is true even on a per–read basis), as has per–read control over captured mapped density (as this is at least δ as a fraction). If desired, one can also control per–read fidelity of final density summed over all q: since the missing density per read is at most $1 – \delta$, then in the worst case (this being the typically extremely unlikely event that all error is concentrated at a single window), every read is missing at most $(1 – \delta)\, r'$ density, where r' is the number of reads q; if this is small, then at every window final density has either high relative accuracy (being no worse than δ) or is small on an absolute scale (being at most $(1 – \delta)\, r'$, which is controllably small). The δ required to achieve this, however, may be extremely close to 1 and the CPU time required for mapping may then be considerable; one might be satisfied with

a theoretically more approximate mapped density that still has excellent quality (such as the global quality guarantees, which typically do not require δ to be anywhere near as close to 1). Even so, the typical exponential fall–off of S' as more and more genomic bases are "wrong" for the current read due to typical fairly tight basecalls implies that even what might seem rather extremely tight error tolerances have a chance at being achieved with practical amounts of computing time. Further, if one can afford for all nodes of T (instead of just leaves) to be augmented with $\mathrm{WGCN_L}$ information, then improved pruning strategies for this application (not detailed here) exist.

The typically huge mapping output files that would otherwise result in this application can be avoided by doing density accumulation directly during the mapping stage. (Lock–free multithreaded accumulation can be achieved with the atomic compare–and–swap operations of modern CPUs, so that all threads can share the same accumulation buffer with low penalty. This is important to conserve RAM, as accumulation buffers are typically substantial in size.) Hence, there is no mapping output until all q in a batch are processed, when mapped density over the entire genome is emitted at once. Further, one can avoid the apparent need to have $\mathrm{WGCN_L}$ expansion information (i.e., the passage from G_n back to W_n) in RAM at mapping time by accumulating density only at representative locations and uniformly distributing from representative locations to all locations in a separate phase after mapping is complete.

The additional applications just sketched illustrate the flexibility of the CokusAlignment framework and how trees T are rich sources of information on reference genomes that can be readily adapted to numerous genomic data analysis tasks.

CokusCalling: Per–Lane, Per–Cycle Gaussian Mixture Models (GMMs) for Basecalls

The most interesting details of CokusCalling have already been described. Unlike alignment to a reference genome, there is no mathematically precise problem to be solved, and so the actual implementation (in MATLAB) is based on empirical experience and is rather "messy," being known to have good performance only on the lanes of interest on the particular flow cells involved in the present project (and it contains a number of parameters and coping strategies to that effect that likely require tuning, replacement, or augmentation for other lanes; while some general strategies have been conceived, more experience is needed to determine the breadth of their applicability and hence they are omitted here). For more detail than contained in the present document, the source files in the distribution available on the web site that accompanies this publication are perhaps as good of a means of communication as any.

However, the QuickTime movies made and used for quality control deserve additional exposition. The distribution available on the web site contains a movie for the example lane `N1try2-L7` used throughout that posting. One can observe a Solexa "blip" or "hiccup" at cycle 22, where a single–cycle increase is observed in Solexa `_sig2` (S_A, S_C, S_G, S_T) signal vectors with at least one negative component, and the fitted GMM components (described further below) are slightly affected. (Even such "minor" phenomena as this were taken as cause for suppression of a cycle from basecalling; there exist cycles of lanes on other flow cells with much more severe problems that were, of course, also suppressed.) The movie readily illustrates some of the behavior of S vectors (which the Solexa software has already corrected for crosstalk and phasing as best it can) with which CokusCalling aims to assist. For example, in the S_G–vs.–S_T plot at the right edge of the middle row (this plot being introduced below along with the eight others), one can see that while at cycle 6 the "G" and "T" concentrations are well–separated (although the angle between them in this two–dimensional projection is only ~65° and not 90°) and lie approximately on their respective canonical axes, by cycle 36

the "G" concentration has rotated considerably in the direction of "T" so that "G" is now rather not aligned along its canonical axis and the separation between it and the "T" concentration in this projection has reduced to ~35° (in addition to both concentrations being fuzzier due to the many repetitions of the "only" near–perfect per–cycle chemistry by that time).

There is one losslessly–compressed (Apple Animation codec) looping 24–bit color 797–by–705 pixel 12 frames–per–second `.mov` per lane. Frames correspond to cycles (generally, 6..36). The main data displayed are the four–dimensional (S_A, S_C, S_G, S_T) signal vectors from the Solexa software `_sig2` files. There is one vector per raw spot in the lane, so that there are typically millions of vectors per lane. However, it is hard to plot four–dimensional data directly and the nature of the Solexa signals does not really require it; instead, nine two–dimensional projections are shown (arrayed into a three–by–three grid). Rightward is increasing on x–axes and upward is increasing on y–axes. Levels $x = 0$ and $y = 0$ are shown as very thin light gray lines.

There are essentially six ways to take S_A, S_C, S_G, and S_T two at a time, and giving one member to the x–axis and the other to the y–axis results in the upper six plots. (For example, the plot in the upper–left corner shows S_A on x and S_C on y; S_G and S_T are dropped from each vector in this plot, or, equivalently, the S_G and S_T axes are orthogonally projected out. The result is that stronger S_A is right, stronger S_C is up, and vectors that are mostly S_G and S_T appear near the origin.)

As it is useful to see all four bases at once, the lower three plots are also included. There are essentially three ways to partition S_A, S_C, S_G, and S_T into two groups of two, and these correspond to the three lower plots. Differences of pairs are plotted. (For example, the plot in the lower left corner has $S_A - S_C$ on x and $S_T - S_G$ on y. Hence, S_A is rightward, S_C is leftward, S_G is downward, and S_T is upward.)

Because there are so many vectors (or, after projection, two–dimensional points) to plot, a density plot rather than a scatter plot is used. The number of points m falling in each bin of a 250–by–250 grid is converted into a color by linearly mapping $\log_{10}(1 + m)$ from $0 \rightarrow 4$ to blue → cyan → green → yellow → orange → red (with values above 4 clipping to the brightest red). Border bins are attached and collect what would otherwise be out–of–range vectors.

Ideally, one would see three stationary red dots in each of the upper six plots (one dot directly on the $+x$–axis, one directly on the $+y$–axis, and one — really an overlapping pair — at the origin), and four stationary red dots in each of the three lower plots (one along $+x$, $-x$, $+y$, and $-y$). Of course, that is too much to expect; a physical measurement process is involved. The next best result one might hope for is small, tight, well–separated circular fuzzy blobs instead of dots, but that is still too much to hope for. Instead, the best one typically observes are medium–sized, decently–separated, narrow elliptical (or, pre–projection, four–dimensional hyperellipsoidal) fuzzy blobs. These are the signal concentrations (one for each call of "A", "C", "G", and "T") that CokusCalling is trying to fit. Distractions include the (sometimes heavy) sheets of density connecting concentrations which result from Solexa–software–called raw spots that actually involve multiple physical DNA clusters.

As expected, signal concentrations tend to get fuzzier (rounder and more overlapping) as cycles go by. They do not appear to shrink much, however, because the plots are auto–scaled to deal with the general decrease in intensity over time. Each axis of the upper six plots runs $[-l/3, +l]$ and each axis of the lower three plots runs $[-l, +l]$, where l is the largest Euclidean norm of the mean vectors of fitted "A", "C", "G", and "T" hyperellipsoids for that cycle.

The outlines of the 97.5 percentile hyperellipsoids of individual fitted mixture components (each of which projects to an ordinary ellipse in each two–dimensional plot) are shown. "A" is in red, "C" is in green, "G" is in blue, and "T" is in violet. Under human inspection, it is generally obvious when the mixture fitter has failed to track the signal concentrations accurately.

For each cycle, mixture probabilities ($p^{(A)}$, $p^{(C)}$, $p^{(G)}$, $p^{(T)}$) from the GMM model for that cycle are collected. The median $m^{(A)}$ of the $p^{(A)}$, $m^{(C)}$ of the $p^{(C)}$, etc. are found, and vector ($m^{(A)}$, $m^{(C)}$, $m^{(G)}$, $m^{(T)}$) is linearly re–scaled so that its components sum to 1. The result is displayed in textual form as modeled percent "A", "C", "G", and "T" at the top. ♦