Supplementary Data

Applications Note

WALT: fast and accurate read mapping for bisulfite sequencing

Haifeng Chen, Andrew D. Smith* and Ting Chen*

1. Spaced Seeds

WALT employs *periodic spaced seed* (010)* originally described by Chen et al. (Chen, et al., 2009). The star ("*") indicates repeating of the *spaced seed pattern* 010 until the spaced seed reaches the desired weight, or the end of a sequence to which it is applied. When the spaced seed is applied to a sequence, the bases aligned to ones ("1") are extracted and concatenated to form a subsequence. As shown in the following example, the subsequence ATTTTGTGTATTATTT is formed when the spaced seed is applied to the sequence.

```
AATATAATTTTGTTAAGGGTGAGTGTTTAGTTGTTTTATTGATTGATT
010010010010010010010010010010010010010010010010
 A   T   T   T   T   G   T   G   T   A   T   T   A   T   T   T
```

WALT applies three periodic spaced seed $(010)^*$, $0(010)^*$ and $00(010)^*$, each with 0, 1, and 2 shifts of the original spaced seed (010)*, respectively, to each read to extract three subsequences. These three subsequences for each read are the three seeds used for mapping, as shown in Table S1.

Table S1. Three spaced seeds used in WALT

(1) 010010010010010010010010010010010010010010010010…
(2) 0010010010010010010010010010010010010010010010001…
(3) 0001001001001001001001001001001001001001001001000…

The *seed weight* for a read is determined by the length of the *third spaced seed* that reaches the end of the read, and the formula is shown in Equation S1,

$$sw = \left\lfloor \frac{rl - 2}{3} \right\rfloor \qquad (S1)$$

where sw is the seed weight and rl is the read length. Figure S1 shows an example of how WALT applies *three spaced seeds* to a read to extract three subsequences, which are three seeds used for mapping.

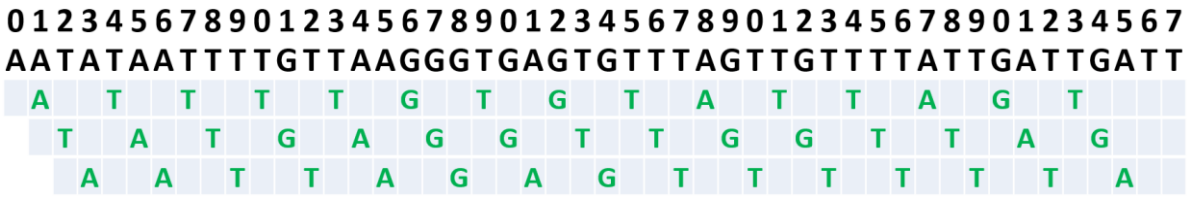

Figure S1. An example to show how WALT obtains three seeds from a read.

WALT preprocesses a reference genome with spaced seed (010)* applied to each genomic position to build an index table, applies three spaced seeds, (010)*, 0(010)*, 00(010)* (each with 0, 1, and 2 shifts to the spaced seed (010)*) to each read, respectively, and searches the index table three times. Therefore, the genomic positions that match the read exactly will be found by the first seed, the genomic positions that have one mismatch to the read will be found by the first or second seed, and the genomic positions that have two mismatches or more to the read will be found by the first, second, or third seed. Based on this observation, we design a simple strategy: once WALT finds an exact match using the first seed, it stops searching because the best unique solution has been found. Otherwise, WALT continues searching with the second seed: if it finds a genomic position with only one mismatch during the first two searches, it stops searching because the best unique solution with one mismatch has been found. Otherwise, WALT combines results from all three searches to find the best solution. In this case, the best solution will have at least two mismatches to the read.

2. Algorithms of WALT

2.1 Building Index Table

Figure S2 illustrates how WALT builds a hash table using a periodic spaced seed (010)*. For every genomic position in the reference genome, WALT applies the spaced seed (010)* to extract a *subsequence* starting from that position. For example, for genomic position 0, the extracted subsequence is TAGATTTTATTAA…, and for genomic position 4, it is AATTTGATATAT…. In this illustration, the first $k = 3$ nucleotides of the subsequences are used as the hash key to index the genomic position in the corresponding hash bucket, but in practice, WALT uses $k = 12$.

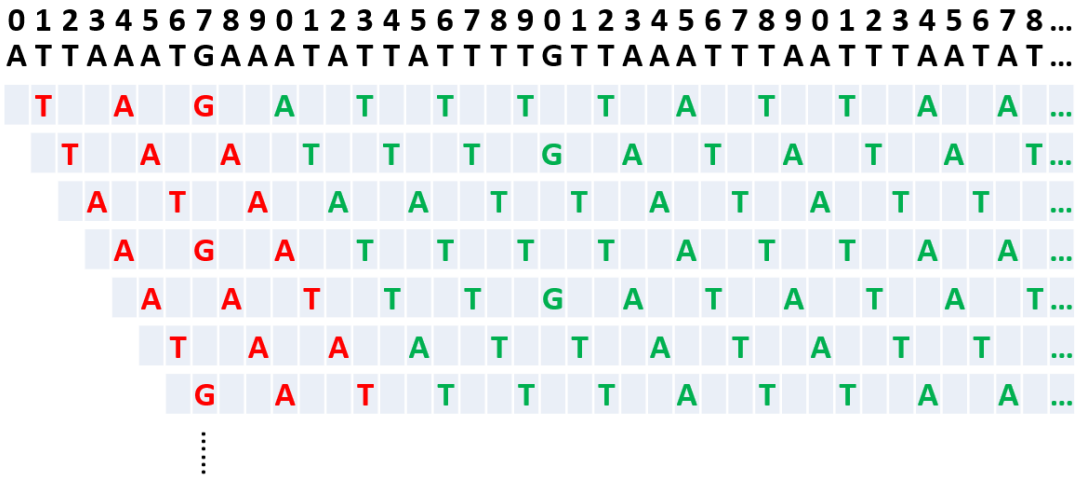

Figure S2. Applying periodic spaced seed (010)* to a genome sequence.

Figure S3 shows the hash table for the reference genome shown in Figure S2. The total number of hash keys is 3^k. In each hash bucket, WALT sorts all candidates (extracted subsequences) in the dictionary order, and applies a binary search to identify exact matches. In practice, only genomic positions are stored in the index table, instead of the actual subsequences. As shown in Figure S3, there are 5 genomic positions, 29, 24, 14, 7 and 9, in the hash bucket with key ATT, and these positions are sorted in dictionary order, ATT, ATTAA, ATTATATT, ATTTGATATAT, ATTTTATTAA, respectively.

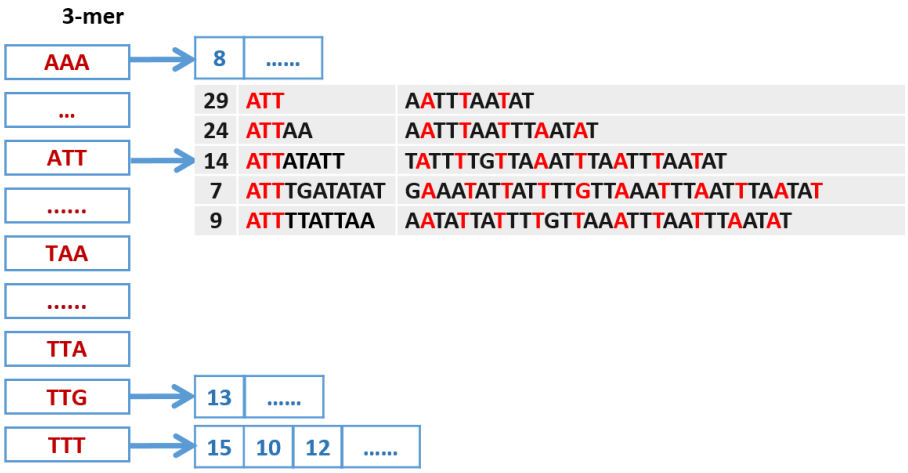

Figure S3. Hash table for the reference genome shown in Figure S2.

2.2 Bisulfite Mapping

WALT uses the first *k* nucleotides in the seed to locate the hash bucket and binary search is applied for nucleotides after the first *k*. As shown in Figure S4, the first seed for the read AATATGATTTTGTTAAATTTAATA is ATTTTATT. The first 3 nucleotides ATT is used as the hash key to locate the hash bucket. The nucleotides after the first 3 in the seed are TTATT. There are 5 positions in the reference genome in the same hash bucket, and they are sorted (Figure S3). WALT applies a binary search to find exact matches for TTATT. As shown in Figure S3, out of 5 positions, only the suffix of the genomic position 9 with spaced subsequence TTATTAA matches TTATT. Then, WALT validates position 9 by aligning the whole read (Figure S4) against the genome sequence (Figure S2) to count the number of mismatches.

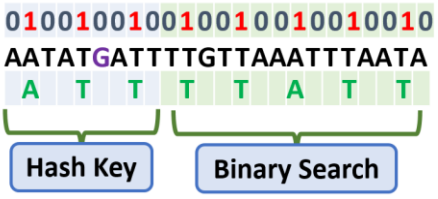

Figure S4. Spaced seed applied to a read and searching the index table.

Since WALT applies three spaced seeds to each read, WALT searches the index table using the three extract subsequences of the read, and computes the number of mismatches.

3. The majority of uniquely mapped reads are located in small hash buckets.

We have observed that genomic positions are not evenly distributed in the hash buckets. Some hash buckets are extremely large, for example, with millions of genomic positions. If a read hits a large bucket, validating these candidate positions takes a large amount of time. Furthermore, such buckets will be hit by millions of reads as each of these candidate positions can be sequenced. Here we discovered that by using periodic spaced seeds $(010)^*$, $0(010)^*$ and $00(010)^*$, the majority of uniquely mapped reads are located in small hash buckets.

Tables S2 and S3 show the accumulated percentage of uniquely mapped reads located in different bucket sizes when considering candidates in all hash buckets. More than 87% of uniquely mapped reads are located in the hash buckets with only one candidate. More than 99% of uniquely mapped reads are located in the hash buckets with less than 5,000 candidates. If we only consider hash buckets with less than 5,000 candidates, we could identify about 99% of uniquely mapped reads.

Table S2. Single-end: accumulated percentage of uniquely mapped reads in different bucket sizes

	WALT-all uniquely mapped reads	1	5	10	50	100	500	1000	5000
SRR1532534	42,259,476	0.877	0.922	0.935	0.958	0.966	0.981	0.985	0.993
SRR948855	41,701,812	0.893	0.932	0.944	0.964	0.971	0.984	0.987	0.995
SRR2296821	15,667,881	0.910	0.967	0.977	0.991	0.996	1.000	1.000	1.000

* WALT-all is the program kept all hash buckets, and WALT is the program only kept the hash buckets with size less than 5,000.

Table S3. Paired-end: accumulated percentage of uniquely mapped reads pair in different bucket sizes

	WALT-all uniquely mapped reads pair		1	5	10	50	100	500	1000	5000
SRR1532534	39,825,389	mate 1	0.863	0.910	0.922	0.946	0.954	0.970	0.976	0.989
		mate 2	0.861	0.909	0.921	0.945	0.953	0.970	0.976	0.988
SRR948855	38,977,964	mate 1	0.885	0.926	0.937	0.957	0.964	0.977	0.981	0.992
		mate 2	0.885	0.926	0.937	0.957	0.965	0.978	0.982	0.992
SRR2296821	15,509,508	mate 1	0.882	0.956	0.969	0.989	0.995	1.000	1.000	1.000
		mate 2	0.882	0.957	0.969	0.989	0.994	1.000	1.000	1.000

Table S4 and S5 show the comparison of runtime and the number of uniquely mapped reads between WALT-all and WALT. WALT-all keeps all candidates in hash buckets, while WALT filters out hash buckets with more than 5,000 candidates. WALT not only is significantly faster than WALT-all, but also obtains almost the same percentage of uniquely mapped reads.

Table S4. Single-end: runtime and percentage of uniquely mapped reads in WALT-all and WALT (hours)

	Runtime		Uniquely mapped reads	
	WALT-all	WALT	WALT-all	WALT
SRR1532534	1.58	0.71	42,259,476 (84.52%)	42,276,826 (84.55%)
SRR948855	1.24	0.85	41,701,812 (83.40%)	41,713,985 (83.43%)
SRR2296821	0.11	0.09	15,667,881 (72.73%)	15,666,896 (72.72%)

* For data sets SRR1532534 and SRR948855, WALT-all obtained a little less uniquely mapped reads than WALT since the second or the third seeds for a few reads are located in the hash bucket with more than 5,000 candidates. If these candidates are filtered out, these reads are uniquely mapped, but these reads maybe ambiguously mapped if considering more candidates.

Table S5. Paired-end: runtime and percentage of uniquely mapped reads pair in WALT-all and WALT (hours)

	Runtime		Uniquely mapped reads pair	
	WALT-all	WALT	WALT-all	WALT
SRR1532534	6.32	2.43	39,825,389 (79.65%)	39,526,217 (79.05%)
SRR948855	4.66	2.61	38,977,964 (77.96%)	38,790,909 (77.58%)
SRR2296821	0.35	0.32	15,509,508 (71.99%)	15,508,389 (71.99%)

4. Comparison of Runtime, Accuracy and Memory Usage

Tables S6~S11 show comparisons of runtime, accuracy and memory usage for WALT, Bismark (Krueger and Andrews, 2011), BSMAP (Li and Li, 2009), BS-Seeker2 (Guo, et al., 2013), BatMeth (Lim, et al., 2012), BWA-meth (Pedersen, et al., 2014). PASS-bis (Campagna, et al., 2013) and segemehl (Otto, et al., 2012) were not included because they did not complete the single-end read mapping within 100 hours for date sets SRR1532534 and SRR948855. MethylCoder (Pedersen, et al., 2011) was superseded by BWA-meth. BRAT-BW (Harris, et al., 2012) did not include read names (id) in the output file, and BRAT-nova (Harris, et al., 2016) did not complete building index for human genome hg19 in 100 hours, so they were not included in comparisons.

As shown in Tables S6-7, for single-end read mapping, WALT is about 6 times faster than BSMAP, 12 times faster than BWA-meth, and more than 35 times faster than Bismark, BS-Seeker2 and BatMeth. For pair-end read mapping, WALT is about 6 times faster than BSMAP, 10 times faster than BWA-meth, 15 times faster than Bismark and more than 35 times faster than BS-Seeker2.

Table S6. Comparison of **runtime** on single-end reads (hours)

	WALT	Bismark	BSMAP	BS-Seeker2	BatMeth	BWA-meth
SRR1532534	0.71	26.85	4.14	39.59	40.87	8.36
SRR948855	0.85	29.13	6.09	42.86	30.13	9.08
SRR2296821	0.09	3.11	0.09	10.32	4.00	2.06

Table S7. Comparison of **runtime** on paired-end reads (hours)

	WALT	Bismark	BSMAP	BS-Seeker2	BatMeth	BWA-meth
SRR1532534	2.43	38.72	12.32	102.51		22.78
SRR948855	2.61	37.10	21.55	91.03		23.40
SRR2296821	0.32	7.46	0.22	21.02		6.09

* BatMath does not support pair-end bisulfite mapping.

As shown in Tables S8-9, WALT, Bismark, BSMAP, BS-Seeker2 and BatMeth have very high recall and precision, while BWA-meth has low recall and precision in paired-end read mapping.

Table S8. Comparison of **accuracy** on single-end read mapping

		WALT	Bismark	BSMAP	BS-Seeker2	BatMeth	BWA-meth
SRR1532534	Recall	0.982	0.973	0.994	0.961	0.990	0.991
	Precision	0.991	0.976	0.988	0.934	0.993	0.852
SRR948855	Recall	0.984	0.974	0.995	0.964	0.990	0.991
	Precision	0.993	0.969	0.991	0.927	0.995	0.839
SRR2296821	Recall	0.987	0.974	0.995	0.965	0.994	0.993
	Precision	0.995	0.978	0.993	0.923	1.000	0.733

Table S9. Comparison of **accuracy** on paired-end read mapping

		WALT	Bismark	BSMAP	BS-Seeker2	BatMeth	BWA-meth
SRR1532534	Recall	0.965	0.968	0.991	0.955		0.495
	Precision	0.984	0.943	0.967	0.886		0.702
SRR948855	Recall	0.971	0.972	0.993	0.960		0.494
	Precision	0.989	0.935	0.973	0.878		0.674
SRR2296821	Recall	0.986	0.959	0.994	0.946		0.489
	Precision	0.996	0.960	0.967	0.862		0.628

Tables S10-11 show the memory usage for each dataset. For human genome (the first two datasets), WALT uses about 15 Gb memory for single-end mapping, and 17 Gb for paired-end mapping.

Table S10. Comparison of **memory** usage on single-end reads (Gb)

	WALT	Bismark	BSMAP	BS-Seeker2	BatMeth	BWA-meth
SRR1532534	14.94	9.55	7.83	11.67	16.55	12.30
SRR948855	15.05	9.58	7.84	11.71	16.55	12.33
SRR2296821	1.02	0.53	1.02	2.52	7.77	2.07

Table S11. Comparison of **memory** usage on paired-end read mapping (Gb)

	WALT	Bismark	BSMAP	BS-Seeker2	BatMeth	BWA-meth
SRR1532534	16.89	9.64	7.88	9.47		12.52
SRR948855	16.95	9.64	7.89	9.72		12.48
SRR2296821	2.89	0.54	1.06	3.25		2.30

5. Runtime on different lengths of reads

Table S12 compares the runtime of WALT, Bismark, BSMAP, BS-Seeker2, BatMeth and BWA-meth for datasets with different read lengths. Six datasets with read length L=50, 60, 70, 80, 90 and 100 respectively were generated by truncating the reads from SRR948855 (Fortin, et al., 2014). Each dataset takes the first L nucleotides from the first 10 million read pairs. WALT runs faster when the read length increases. It is because spaced seeds for longer reads have heavier weights, which help to reduce the number of random hits.

Table S12. Runtime on different read length (hours)

	Read Length	50	60	70	80	90	100
Single-end	WALT	0.71	0.27	0.21	0.19	0.18	0.18
	Bismark	1.90	2.57	2.84	3.32	4.30	5.61
	BSMAP	0.33	0.46	0.65	0.43	0.62	1.03
	BS-Seeker2	7.14	7.36	7.75	8.27	8.59	9.09
	BatMeth	46.52	21.47	15.06	11.04	8.46	6.39
	BWA-meth	1.55	1.50	1.52	1.61	1.71	1.88
Paired-end	WALT	2.66	1.20	0.85	0.63	0.59	0.61
	Bismark	4.60	5.50	6.63	6.69	7.82	8.09
	BSMAP	0.62	1.11	1.79	1.40	2.09	3.08
	BS-Seeker2	20.98	23.88	22.83	20.88	24.04	22.43
	BatMeth						
	BWA-meth	5.38	4.47	4.47	4.62	4.93	5.48

6. Uniquely mapped reads on different number of mismatches in WALT

Table S13. The number and percentage of uniquely mapped reads on different number of mismatches in WALT

Number of mismatches	SRR1532534		SRR948855		SRR2296821	
	Single-end	Paired-end	Single-end	Paired-end	Single-end	Paired-end
0	34,689,403 (69.38%)	27,200,475 (54.40%)	34,743,554 (69.49%)	27,559,957 (55.12%)	13,728,491 (63.72%)	12,697,635 (58.94%)
1	5,489,829 (10.98%)	8,090,762 (16.18%)	5,211,452 (10.42%)	7,388,497 (14.78%)	1,279,528 (5.94%)	1,809,749 (8.40%)
2	1,131,020 (2.26%)	2,394,739 (4.79%)	898,452 (1.80%)	2,342,240 (4.68%)	287,268 (1.33%)	473,105 (2.20%)
3	448,474 (0.90%)	983,247 (1.97%)	361,270 (0.72%)	810,523 (1.62%)	178,251 (0.83%)	190,001 (0.88%)
4	250,467 (0.50%)	511,419 (1.02%)	228,968 (0.46%)	404,315 (0.81%)	107,945 (0.50%)	161,688 (0.75%)
5	155,276 (0.31%)	289,866 (0.58%)	156,000 (0.31%)	238,442 (0.48%)	56,045 (0.26%)	128,729 (0.60%)
6	112,357 (0.22%)	55,709 (0.11%)	114,289 (0.23%)	46,935 (0.09%)	29,368 (0.14%)	47,482 (0.22%)
SUM	42,276,826 (84.55%)	39,526,217 (79.05%)	41,713,985 (83.43%)	38,790,909 (77.58%)	15,666,896 (72.72%)	15,508,389 (71.99%)

7. WALT Output Format

WALT supports Tab-delimited SAM (Li, et al., 2009) and MR (Song, et al., 2013) output formats. In WALT, SAM and MR formats have 12 and 8 mandatory fields, respectively. One mapping result has one line and different fields are separated by a Tab. Here shows the mandatory fields in order for SAM and MR formats.

SAM Format

- QNAME (read name)
- FLAG (bitwise FLAG)
- RNAME (chromosome name)
- POS (start position, 1-based, 0 means unmapped)
- MAPQ (255 in WALT)
- CIGAR (CIGAR string)
- RNEXT (chromosome name of the mate read)
- PNEXT (start position of the mate read)
- TLEN (observed segment length)
- SEQ (read sequence)
- QUAL (quality sequence)
- NM-tag (number of mismatches)

MR Format

- RNAME (chromosome name)
- SPOS (start position, 0-based)
- EPOS (end position, 0-based)
- QNAME (read name)
- MISMATCH (number of mismatches)
- STRAND (forward or reverse strand)
- SEQ (read sequence)
- QUAL (qualify sequence)

8. Test Environment and Command Lines

All tests were run on the USC HPC cluster with Intel Xeon E5-2600 processors at clock speed 2.4 GHz. All experiments were run in single thread on single processor. WALT 1.0, Bismark 0.13.1, BSMAP 2.88, BS-Seeker2 2.0.10, BatMeth 1.04 and BWA-meth 0.10 were used in these experiments. Here shows the command lines for them.

WALT single-end mapping

```
walt -i genome.dbindex -r reads_file_1.fastq -o output_single.sam
```

WALT paired-end mapping

```
walt -i genome.dbindex -1 reads_file_1.fastq -2 reads_file_2.fastq -o output_paired.sam
```

Bismark single-end mapping

```
bismark -N 1 -L 32 --path_to_bowtie bowtie2-2.2.4 --bowtie2 gnome_files reads_file _1.fastq
```

Bismark paired-end mapping

```
bismark -N 1 -L 32 --path_to_bowtie bowtie2-2.2.4 --bowtie2 gnome_files reads_file _1.fastq -2
reads_file_2
```

BSMAP single-end mapping

```
bsmap -d gneome_files -a reads_file_1.fastq -p 1 -v 6 -r 0 -o output_single.sam
```

BSMAP paired-end mapping

```
bsmap -d gneome_files -a reads_file_1.fastq -b reads_file_2.fastq -p 1 -v 6 -r 0 -o output_paired.sam
```

BS-Seeker2 single-end mapping

```
./bs_seeker2-align.py --aligner=bowtie2 -g genome_files -m 6 --bt2-p 1 -d
./bs_utils/reference_genomes/ -f sam -i read_file_1 -o output_single.sam
```

BS-Seeker2 paired-end mapping

```
./bs_seeker2-align.py --aligner=bowtie2 -g genome_files -m 6 --bt2-p 1 -d
./bs_utils/reference_genomes/ -f sam -1 read_file_1 -2 read_file_2 -o output_paired.sam
```

BatMeth single-end mapping

```
./batmeth  -g genome_files -i read_file_1 -n 6 -p 1 -o output_single.out
```

BWA-meth single-end mapping

```
./bwameth.py --reference genome_files read_file_1 -t 1 --prefix PREFIX_SINGLE
```

BWA-meth paired-end mapping

```
./bwameth.py --reference genome_files read_file_1 read_file_2 -t 1 --prefix PREFIX_PAIR
```

References

Campagna, D.*, et al.* PASS-bis: a bisulfite aligner suitable for whole methylome analysis of Illumina and SOLiD reads. *Bioinformatics* 2013;29(2):268-270.

Chen, Y., Souaiaia, T. and Chen, T. PerM: efficient mapping of short sequencing reads with periodic full sensitive spaced seeds. *Bioinformatics* 2009;25(19):2514-2521.

Fortin, J.P.*, et al.* Functional normalization of 450k methylation array data improves replication in large cancer studies. *Genome Biol* 2014;15(12):503.

Guo, W.*, et al.* BS-Seeker2: a versatile aligning pipeline for bisulfite sequencing data. *BMC Genomics* 2013;14:774.

Harris, E.Y., Ounit, R. and Lonardi, S. BRAT-nova: fast and accurate mapping of bisulfite-treated reads. *Bioinformatics* 2016.

Harris, E.Y.*, et al.* BRAT-BW: efficient and accurate mapping of bisulfite-treated reads. *Bioinformatics* 2012;28(13):1795-1796.

Krueger, F. and Andrews, S.R. Bismark: a flexible aligner and methylation caller for Bisulfite-Seq applications. *Bioinformatics* 2011;27(11):1571-1572.

Li, H.*, et al.* The Sequence Alignment/Map format and SAMtools. *Bioinformatics* 2009;25(16):2078-2079.

Li, Y.X. and Li, W. BSMAP: whole genome bisulfite sequence MAPping program. *Bmc Bioinformatics* 2009;10.

Lim, J.Q.*, et al.* BatMeth: improved mapper for bisulfite sequencing reads on DNA methylation. *Genome Biol* 2012;13(10):R82.

Otto, C., Stadler, P.F. and Hoffmann, S. Fast and sensitive mapping of bisulfite-treated sequencing data. *Bioinformatics* 2012;28(13):1698-1704.

Pedersen, B.*, et al.* MethylCoder: software pipeline for bisulfite-treated sequences. *Bioinformatics* 2011;27(17):2435-2436.

Pedersen, B.S.*, et al.* Fast and accurate alignment of long bisulfite-seq reads. In, *ArXiv e-prints*. 2014.

Song, Q.*, et al.* A Reference Methylome Database and Analysis Pipeline to Facilitate Integrative and Comparative Epigenomics. *Plos One* 2013;8(12).