This is a review of Heng Li “Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM”, v2, available from http://arxiv.org/abs/1303.3997
The review was written by Aaron Darling and presented at the Bioinformatics Lunch Bytes meeting at the University of Technology Sydney.

Review of the manuscript:

The manuscript introduces the “mem” module of bwa, a read mapping and alignment program that is widely used in the high throughput sequence analysis field. The manuscript is written in the bioinformatics app note format, which has a 2 page limit. Even though the page limit is highly restrictive the manuscript manages to include some helpful introduction and discussion. I found the methods section to be lacking important detail though (see below).

Major comments:

1. Why does the reseeding approach work? The manuscript does not give any insight into why this helps improve accuracy. The issue seems to be that the longest matches (the SMEM) for a query sequence might not be part of the true alignment. By allowing shorter matches to be alignment seeds it can still find the true alignment in the occasional case where the longest MEM matches the wrong place. That makes sense, but why require higher multiplicity and restrict only to matches covering the middle base in the SMEM? Also, why is the reseeding threshold 28nt? I can wager a guess that this seemed like a good speed/accuracy tradeoff on a particular dataset but it would be nice to have some idea of whether this is a good heuristic for a wide range of datasets or if it is really tuned for, e.g. human genomes. Sequence analysis has too many magic numbers already. Finally, how frequently is the SMEM not part of the true chain in real data? One way to evaluate this would be to test accuracy with the reseeding disabled.
2. How does the seed ranking work? The manuscript says “We rank a seed by length of chain it belongs to and then by the seed length” but this plain language description lacks the precision required to reproduce the method. Is chain length defined as the span of the chain in the query, the reference sequence, the sum of constituent seed lengths or something else? This should be explained.
3. The program evaluation is quite limited, in that only a single substitution & indel rate is used to compare aligners. Comparing a wide range of sequence divergences is a pretty substantial computational undertaking so it’s ok to keep the benchmarking simple, but the interpretation of results needs to be strongly qualified in the manuscript. I am specifically concerned that the SMEM based alignment seeding strategy will fall apart at higher substitution or indel error rates because it depends on finding long exact matches. What happens to reads with dense polymorphisms or sequencing data generated on instruments with high error rates such as the Pacific Biosciences instrument?
4. There is one glaring omission in the comparison set of aligners and that is LAST (http://last.cbrc.jp). LAST can do some clever things such as quality-aware gapped alignment scoring and approximate Bayesian aligned read pairing, in addition to long sequence alignment. If there is some reason that a comparison to LAST isn’t possible or

appropriate it should be stated so rather than leaving us to wonder about it.

Minor comments:

1. There are numerous minor grammatical and spelling errors which could be easily fixed and would improve readability
2. There seems to be a single mismatch penalty used for the gapped alignment, however, some nucleotide substitutions happen much more frequently than others. At a minimum the ability to treat transitions differently than transversions would be nice, though being able to use a 4x4 substitution matrix (and/or estimating it directly from the data) would be ideal.
3. No mention is made of whether or how the per-base quality estimates produced by sequencing instruments are used during gapped alignment.
4. The value of the “SW rescuing” approach is unclear. The manuscript says the 2nd best SW score is recorded -- what happens when there are many equally good alignments? The alignment with the 2nd best score might be minimally different. Or is this referring to the best alignment to a *different* region of the reference sequence? If so, the explanation needs to be improved.
5. What is a “standing seed” in section 3 paragraph 2?
6. The discussion comment that “only bwa mem scales well to large genomes” is misleading since there is a well developed body of literature on methods and software for pairwise and multiple genome alignment covering large genomes and some of them scale very well to large genomes. bwa mem and nucmer are not the only two programs in this space.

Other comments:

1. The description of how pairing scores are calculated was pleasantly understandable (to me, anyway) and succinct. One question: if each read in a pair has many possible alignment locations then a large number of possible combinations arise. Are all of these scored or does bwa bound the search somehow?
2. “is more performant” -> performs better

Review of the software:

I obtained bwa-0.7.5a from http://sourceforge.net/projects/bio-bwa/files/
After downloading I used the following commands to build and run the program:

```
tar xjf bwa-0.7.5a.tar.bz2
cd bwa-0.7.5a/
make
cp bwa $HOME/bin/
```

Running the software with no arguments produces a help message with enough detail to get started, and running the mem subcommand without arguments gives all the options for bwa mem alignment. I was pleased to see the fine grain control over the alignment parameters

available for power users, though I'm sure 99% of users will stick with defaults here because it can be very hard to determine sensible settings for these parameters.
One parameter question: the manuscript describes a *Z* dropoff which is like BLAST's X-drop but allows the gap in either the reference or query to be penalty-free. The program help for bwa mem has:

```
-d INT      off-diagonal X-dropoff [100]
```

Is this the same thing? If so it would help to improve the variable naming consistency.

Running bwa:

```
bwa index C_botulinum_F_Langeland.fasta
bwa mem C_botulinum_F_Langeland.fasta cbot_mem1.fq cbot_mem2.fq > cbot_mem.sam
```

Based on my comments above, I decided to run my own comparison of bwa to lastal with the approximate Bayesian paired read mapping module. To do this I simulated paired end reads from the *Clostridium botulinum* F Langeland genome with artsim:

```
art_illumina --paired -l 100 -f 1 -m 400 -s 50 -i
C_botulinum_F_Langeland.fasta -o cbot_mem
```

The above command generates about 1x simulated coverage of the *C. botulinum* reference. I then ran bwa mem as above. last-193 was run as (paramters taken verbatim from their documentation at http://last.cbrc.jp/doc/last-pair-probs.html):

```
time lastal -i1 -e120 -Q1 cbot_last cbot_mem1.fq > cbot_lastal1.maf
time lastal -i1 -e120 -Q1 cbot_last cbot_mem2.fq > cbot_lastal2.maf
time last-pair-probs.py cbot_lastal1.maf cbot_lastal2.maf > lastal_pairs.maf
time maf-convert.py sam lastal_pairs.maf > lastal_pairs.sam
```

accuracy was evaluated using a short perl script (see file bwa_mem_eval.tar.bz2 in figshare):

```
./accuracy.pl cbot_mem.sam < cbot_mem.both.aln
precision: 0.992127563556135   recall: 0.998284561049445
./accuracy.pl lastal_pairs.sam < cbot_mem.both.aln
precision: 1       recall: 0.988837162737148
```

So lastal has remarkably good precision but aligns slightly fewer of the reads with the recommended settings than bwa mem. If we turn these numbers into an F1 score (http://en.wikipedia.org/wiki/F1_score) we get lastal: 0.98883 bwa-mem: 0.99519 so if you like the balance of precision and recall provided by F1, bwa-mem seems like a good choice. For things like identifying rare variants, I will probably stick with lastal when possible since the occasional misalignments could potentially end up creating a lot of rare variants. It might also be possible to improve lastal's recall with a lower e-value threshold, I did not explore this.

One difficulty I encountered in benchmarking these is an apparent bug in artsim where it reports

the location of one read in each pair seemingly at random, while the other read's location was given correctly. I introduced some slop into the accuracy script to work around this, in particular that an alignment would be called correct if the mapped location was within 550nt of the true location.

Review of the code:
The program is implemented in C, with a minimalist but functional Makefile provided. I only evaluated the bwamem part of the code. The header file bwamem.h is well documented, with understandable comments describing what the variables, structures, and (most) function interfaces are doing. Use of doxygen structured documentation is nice. The implementation in bwamem.c has much less code commenting. Superficially it seems relatively modular with sensible enough function and variable names that a seasoned and motivated C coder could understand and modify it. I didn't notice any obvious coding faux pas such as large copied blocks of code, although a few smaller instances could be refactored into functions to remove duplication. In a thorough code review I would probably suggest denser commenting and unpacking or at least description of gems like this:

```
if (which && ((p->cigar[p->n_cigar-1]&0xf) == 4 ||
(p->cigar[p->n_cigar-1]&0xf) == 3)) qb += p->cigar[p->n_cigar-1]>>4;
```

One little annoyance is the use of hard-coded hexadecimal constants in `mem_aln2sam()`. While their meaning may be obvious to the creator of the SAM format it is not very readable for the rest of us. On the whole though it looks pretty good. Better than the code I write (though I can not claim to be a model coder).