Supporting Information

Gnerre et al. 10.1073/pnas.1017351108

SI Materials and Methods

Illumina Sample Prep Modifications (Brief Description). Construction of fragment libraries followed standard protocols for shearing of genomic DNA, end repair, adapter ligation, and size selection. The enrichment PCR was performed using AccuPrime Taq DNA polymerase High Fidelity (Invitrogen) with the following cycling conditions: initial denaturation for 3 min at 98 °C, followed by 10 cycles of denaturation for 80 s at 98 °C, annealing and extension for 90 s at 65 °C, and a final extension for 10 min at 65 °C.

Fosmid Jumping Methods (Brief Description, with Details in the Following Paragraphs). Both methods make a 40-kb insert library in a Fosmid vector (1), which is transfected into *Escherichia coli*. After growth, Fosmid DNA is purified from the host. In **Sh**earing **A**nd **R**ecircularization after **C**loning (ShARC), the DNA is sheared and then size selected and recircularized, so that fragments bearing both ends of the 40-kb insert, yet still of a size suitable for Illumina sequencing, can be recovered by amplification with PCR primers derived from the Fosmid vector ends. In **Fos**mids for the explicit purpose of paired-end sequencing by **Ill**umina (Fosill), a site-specific nick is introduced near the cloning site in the Fosmid vector and then propagated by nick translation into the insert, where it is enzymatically cleaved to a double-strand break. Recircularization of the resulting molecule followed by PCR (as in ShARC) delivers appropriately sized products bearing end sequences for Illumina sequencing.

ShARC. Fosmid libraries were constructed as follows, using the pEpiFOS-5 Fosmid vector (EpiCentre Biotechnologies). Genomic DNA (20 μg) was sheared with the HydroShear device (GeneMachines) and end-repaired. Size-selected (35- to 45-kb) fragments were blunt-end ligated to into the pEpiFOS-5 vector. Primary packaged Fosmid libraries were transfected into *E. coli* DH10B, spread at high density on LB plates containing 25 mg/mL chloramphenicol and 5% sucrose, and incubated at 37 °C for 18 h to select for the Fosmid clones. Colonies were then scraped, collected, and purified via a Plasmid Mega Kit (Qiagen). Fosmids were then sheared by HydroShear to an average size of 9 kb to produce a subset of fragments containing the vector backbone and several hundred bases of the genomic insert on either side. The sheared fragments were concentrated with QIAquick columns, gel size selected to 8–10 kb, purified, end-repaired, phosphorylated, and cleaned up using SPRI (Solid Phase Reversible Immobilization) beads. The size of the remaining fragments was assayed using an Agilent 2100 Bioanalyzer. Next, large-scale recircularization was carried out with 6 μg of fragment in a 3-mL reaction containing 100 μL of T3 DNA ligase in a 1× rapid ligation buffer (NEB). The cleaned-up, eluted ligation products (120 μL) were treated with 15 units of DNase (Epicentre) to remove linear DNA. Circularized fragments containing both Fosmid ends were selected by PCR using primers derived from the pEpiFOS-5 sequences adjacent to the insert, which were tailed with Illumina paired-end adapter sequences. The PCR product was then gel size selected to 500–1,000 bp, enriched using standard Illumina paired-end primers, and sequenced.

Fosill. The pFOS1 vector (1) was modified to include an Eco72I blunt-end cloning site flanked by Illumina paired-end sequencing primer sequences and two Nb.BbvC1 nicking endonuclease sites oriented such that both Nb.BbvC1 nicks can be enzymatically translated by DNA polymerase I into the cloned insert. Genomic DNA (30 μg) was sheared and end-repaired. Size-selected (35- to 45-kb) fragments were ligated to AatII and Eco72I double-digested vector arms. Primary packaged Fosill libraries were transfected into *E. coli* DH10B and amplified by overnight growth at 30 °C in 750 mL 2× YT broth containing 15 μg/μL chloramphenicol. Ten micrograms of purified Fosmid DNA (Qiagen QIAfilter plasmid mega purification kit) was nicked by digestion with Nb.BbvC1. The nicks were then translated by a 45-min incubation at 0 °C with DNA polymerase I and dNTPs and then cleaved by digestion with nuclease S1. End-repaired fragments (~100 ng) were circularized in a 500-μL ligation reaction with T4 DNA ligase. The coligated jumping fragments were then PCR amplified by inverse PCR out of the vector, using standard Illumina paired-end enrichment primers. The PCR product was size selected to 400–600 bp and sequenced.

Sequencing was carried out according to the manufacturer's recommended protocols, except as described above.

ALLPATHS-LG Algorithms. We refer to refs. 2 and 3 for background on ALLPATHS and briefly recall its design here. ALLPATHS first corrects errors in reads and then builds unipaths (2):

> For given minimum overlap K, a *branch* in a genome is a place where there is a sequence of K bases (K-mer) that appears in two or more places and for which the next (or previous) bases are different. By breaking the genome at every branch, we decompose it into a collection of sequences that we call *unipaths*. These unipaths form the edges of a *sequence graph*, by which we mean a directed graph whose edges are sequences (4). In fact, the unipath graph is the best possible assembly of the genome from reads of length $K + 1$, achievable in theory given infinite coverage by perfect reads, which contain all genomic ($K + 1$)-mers and hence reveal all branches.

Using read depth, ALLPATHS estimates the copy number of unipaths. Then it chooses "seed" unipaths, that are ideally long, of low copy number, and separated from each other by several kilobases. Around each seed it builds a neighborhood, consisting of unipaths that are iteratively linked to the seed by paired ends and within ~10 kb of it. Then it adds reads to the neighborhood by reaching out from the given unipaths using paired ends. Within each neighborhood, paired ends are then chosen (generally from jumping libraries), and all paths from one end of the pair to the other are found. These paths are glued together to form a graph, which is the neighborhood (local) assembly. These local assemblies are then glued together to form a global assembly graph. Contigs are extracted from this global assembly and then formed into scaffolds, which constitute the final assembly.

Below we describe some of the changes to ALLPATHS that are new to ALLPATHS-LG.

Error correction of reads from fragment libraries. The goal of error correction in ALLPATHS-LG is to correct as many sequencing errors as possible, while introducing as few new errors as possible. Although this process leads to a simplification of the de Bruijn graph that describes the data, our primary goal is not to simplify the graph, but rather to make it more correct. Therefore, unlike in Euler (4) and Velvet (5), we do not try to remove single-base biological differences such as SNPs.

Error correction of reads from sheared jumping libraries is discussed separately (below). Here we describe the ALLPATHS-LG process for error correction of reads from fragment libraries. It is done in two parts by the modules PreCorrect and FindErrors. In both modules, stacks of reads are built on the basis of sharing of a 24-mer. Then base errors are identified and corrected on the basis of frequencies and quality scores of base calls in columns of the stack. This algorithm is described in more detail below. The FindErrors module uses a contiguous 24-mer whereas

PreCorrect uses a split 24-mer (a 25-mer with an unspecified central base). PreCorrect corrects only the central column in the stack. It is run before FindErrors and can correct some errors not found by FindErrors. For example, a read having an error every 23 bases could not be corrected by FindErrors. FindErrors is run twice in succession to increase the number of errors that are corrected. This procedure helps because each base error in a read effectively removes the read from many read stacks, as the number of valid 24-mers is reduced. If a read has more than a few errors it will be harder to correct on the first pass, as the number of valid stacks it belongs to is small. Once some corrections are in place, the read will be present in more stacks and further errors can be identified and corrected.

The chosen K-mer size of 24 strikes a balance, as follows. The larger K is, the more likely a K-mer is to be unique in the genome and hence give rise to a "pure" stack, i.e., one containing reads from only one locus on the genome. Conversely, decreasing K increases the sensitivity of error correction by allowing the correction of reads having more errors.

The error correction algorithm is divided into three phases: the recommendation phase, where a set of recommended changes is compiled; the correction phase, where these changes are considered and possibly carried out; and finally the screening phase, where reads that (after error correction) contain unique 24-mers are discarded. The recommendation set is computed by constructing, for each 24-mer in the reads, the aligned stack of all of the reads containing that 24-mer. Stacks of fewer than six reads are ignored. Then, for the central base column in PreCorrect and for all of the base columns off the 24-mer in FindErrors, the quality scores are summed separately for each of the four potential calls. The base call with the highest quality score sum is declared the winner, but only if the sum is ≥60. Any other base call, with no more than one call of quality 20 or more and quality score sum less than one-quarter that of the winner, is declared a loser call. We note that a SNP supported by two Q20 bases would be protected from correction by this rule. If there is a winner call, a correction recommendation is issued for each of the loser calls. All of the recommendations for all of the reads are collected before moving on to the correction phase. In PreCorrect, the algorithm looks only at the aligned column of bases in the center of the 25-mer, and hence there is at most one recommendation per read base. Here, corrections in a read are made only if they are at a distance of >12 bases from each other (i.e., a correction is not applied when the flanking 12-mers that built the stack are themselves suspect). In FindErrors, because each read appears in as many reads stacks as its number of K-mers, there are, in general, a set of recommendations associated with any read base. Here corrections are made if and only if all of the various correction recommendations for the same base agree with each other. This conservative consensus mechanism is designed to minimize the number of false positives in the error correction. Whenever a correction is adopted, the associated quality score for that base is set to 0.

Fragment pair filling. There are significant advantages to using a larger K-mer size in the assembly process, but as K approaches the read length, there is danger of losing connectivity among the reads, as there is lower probability that reads will overlap by K bases. So that we can use a larger K, we require as input a library of paired reads from fragments whose size is slightly less than twice the read length and then attempt to fill in any gap in sequence between the ends of a given pair. This method allows us to treat the entire "filled fragment" as a single long "superread." This process is carried out after error correction.

We note that a pair of sufficiently long and accurate reads from appropriately sized fragments could be directly joined to each other along their overlap. The longer the reads are, the more viable such a strategy becomes. For 100 base reads, such a direct joining strategy is impractical, because to avoid wrong joins, the overlap between the two reads would need to be reasonably long (say 20 bases), and moreover to avoid collapsing tandem repeats we would need to be confident that the reads do overlap by this much. For example, if fragments were normally distributed as 130 ± 25, then ∼2.5% of pairs would overlap by <20 bases, and worse, the mean superread length would be only 130 bases. With a tighter size distribution one could increase the mean, but such tight distributions are difficult to achieve in the laboratory.

Instead, we use a third read from the entire set of fragment reads to join a given pair together. This method increases the likelihood that the join is correct. To further reduce the incidence of error, we look not just for a single read, but for a second fragment read pair in which one of the reads aligns perfectly across the gap or overlap in the first pair (Fig. S1*A*). To be joined, the patching read must overlap perfectly with each of the two reads by at least 24 bases. It must also have only one possible alignment across the gap, and the partner of the patching read must also align perfectly, at the appropriate distance, to the second pair. The patching read either confirms the overlap in the original pair or, in the case of a gap, provides the missing bases.

More than one valid patching read may be found, and usually they all result in the same filled fragment. Occasionally alternative joins are found and all are considered valid—i.e., the same fragment pair can result in more than one filled fragment (Fig. S1*B*). We note that the fragment filling conditions are quite stringent. The process has a cleansing effect on the data as it tends to cull out pairs that still have errors after error correction. Typically 70–75% of pairs that survive error correction are filled, and of these ∼0.05% have multiple closures.

Unipath creation. This method is as in ref. 3; however, now we take as input to the process the filled fragment pairs rather than the error-corrected reads.

Error correction of reads from sheared jumping libraries. In reads from jumping libraries that are created by blunt-end self-ligation of long fragments, followed by shearing, such as those made using the Illumina protocol, the circularization junction may appear within a read. Such a read is chimeric, containing two parts, separated on the genome by somewhat less than the fragment length (e.g., ∼2 kb). Our strategy is to identify this junction point and then trim off the part of the read to its right, although these bases could in principle be exploited by the algorithm.

We note that because of the junction points, sheared jumping reads cannot be error corrected in the same manner as reads from fragment pairs. Instead ALLPATHS-LG has a combined process that both corrects errors and trims back after junction points. This process works by aligning the sheared jumping reads to the unipaths. By seeding on the first 12-mer in each read, we define candidate alignments. This is a viable strategy because the first 12 bases in a given read are nearly always correct and, because the unipath graph collapses repeats, a given read will in general have a relatively small number of candidate alignments, although usually more than one.

Each candidate alignment is then extended, following the graph where the end of a unipath is reached, and not allowing gaps. During the extension process, we apply a sliding-window error threshold to identify a possible junction point within the read: As we move forward along the read, if we find more than n base mismatches in a window of m bases between the read and unipath, we back off b bases before the first error in the window and call this the "trusted length" of this alignment. These are tunable parameters; the values used in ALLPATHS-LG are $n = 3$, $m = 8$, and $b = 3$. We compute the maximum of these trusted lengths, reasoning that the junction point (if any) should lie beyond it.

Each candidate alignment is then scored by summing the base quality scores of any mismatches along the maximum trusted length. The best scoring alignment is selected, but only if it is better than the next best score by a significant amount; here, we

used a quality score difference of 20, yielding in principle a ~99% confidence in choosing the winner. We then replace the entire read, including the section beyond the junction point, with the unipath sequence it aligned to—thus removing the invalid sequence and error correcting at the same time.

Gap patching (Fig. S3). ALLPATHS-LG patches gaps at two points, after generation of initial unipaths, and after generation of initial scaffolds. These two processes are similar, and we outline here the key steps of the underlying common method (the steps reflect the key difficulty of the problem, which is that gaps are often associated with low coverage and/or low quality sequence, and thus it may be only barely possible to cross them): (*i*) Reads are placed on the unipaths or contigs (in the case of scaffolds). (*ii*) In the case of scaffolds, there is already a natural order on the contigs. For unipaths, we use the read placements of step *i* to define a putative order of the unipaths, in effect scaffolding them. (*iii*) Each consecutive pair of unipaths or contigs defines a gap. (*iv*) The read placements of step *i* allow us to define a pool of oriented reads that might land in a given gap. (*v*) A first alignment of reads to each other is performed. For each gap, we seed on 12-mer perfect matches to find gap-free alignments between reads in the pool, discarding those alignments having an error rate >20% or whose aligned portion is <40 bases. (*vi*) Errors in reads are corrected. For each read in the pool, we form the stack of reads that are aligned to it. We traverse the columns of this stack, allowing the bases in the column to vote according to their quality scores. So long as the runner-up base has score less than a fixed constant (200), we change the base on the read. (*vii*) To cleanse the reads, we find all of the 16-mers *S* that could be party to a bridge across the gap and then trim each read from the left and the right so that it contains only 16-mers from *S*. (*viii*) The reads are then aligned to each other. We find all perfect overlaps of length ≥15 between the reads. (*ix*) To identify closures, we try to cross the gap by walking along perfect overlaps between the reads. Multiple closures are allowed; however, if >10 are found, the gap closing operation is aborted. The computation is bounded by an internal counter to prevent "blowing up." (*x*) In the unipath case, all closures are incorporated into the unipath graph. In the scaffold case, closures outside the predicted gap size are excluded, and the remaining closures are merged into a common sequence that may contain mismatch ambiguities, thus providing a single closure of the gap with some ambiguities labeled and in some cases new errors. Sequence introduced in patches during the scaffolding step typically comprises ~2% of the assembly.

Flattening. As in ALLPATHS, the ALLPATHS-LG assembly process yields an assembly graph, which can be locally complicated, and this graph is then turned into scaffolds that are inherently linear. To do this, ALLPATHS 2 defined a contig break at each branch point in the graph. This method worked on bacteria but would not work well on polymorphic genomes in which branch points occur at great frequency. Therefore in ALLPATHS-LG the graph goes through an initial flattening phase in which branches arising from single-base mismatches are converted into ambiguous base codes. This phase does not lose information; however, ALLPATHS-LG also flattens some other features including short indels, and this flattening does result in information loss. In some cases an error is introduced or an allele is lost. A better approach would retain this information and represent it as ambiguities in the final assembly.

Scaffolding. In the scaffolding process, we use read pair data from jumping libraries to connect assembled contigs into scaffolds. We found that the greatest challenge in scaffolding the data of this work originated from artifacts present in the read pair data. First consider the fragment size distribution for a given library, which we compute by aligning read pairs to unipaths. It is approximately Gaussian, but often has asymmetry and/or secondary peaks. Second, in the construction of sheared jumping libraries, after shearing of circles, some fragments are selected that do not contain the ligation junction point, resulting in read pairs having reverse orientation and short separation (Fig. S2*E*).

We address both of these issues by using a more robust statistical approach to deal with the inconsistencies in linkage evidence from individual read pairs. First, a link between two scaffolds is defined by a read pair, with each read aligned to one of the scaffolds. The alignment and separation statistics of paired reads (mean and SD) imply the orientation and gap size between the scaffolds. Next we define a bundle to be a subset of links inducing consistent orientations and gaps between two scaffolds. We cluster the links into bundles (details omitted) and then score the bundles as follows: given a bundle of n links $\{\mu_i, \sigma_i\}_{i=1}^n$, where μ_i and σ_i denote the gap size and SD implied by a link, we compute an expected gap size and standard deviation (μ, σ) between scaffolds by $\frac{1}{\sigma^2} = \sum_{i=1}^n \frac{1}{\sigma_i^2}$ and $\frac{\mu}{\sigma^2} = \sum_{i=1}^n \frac{\mu_i}{\sigma_i^2}$. We assign a score s to each bundle by

$$s\left(\{\mu_i, \sigma_i\}_{i=1}^n\right) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(\frac{\mu_i - \mu}{\sigma_i}\right)^2}.$$

This definition is designed to provide an estimate of how far from normal the distribution of the n implied gap sizes in the bundle truly is ($s = 1$ if the distribution of the gaps $\{\mu_i, \sigma_i\}_{i=1}^n$ is a perfect Gaussian). Then we define the weight of a bundle by

$$w\left(\{\mu_i, \sigma_i\}_{i=1}^n\right) = \sqrt{n}\cdot\exp\left\{-n\cdot s\left(\{\mu_i, \sigma_i\}_{i=1}^n\right)\right\}.$$

Finally we choose the heaviest bundle to imply a final orientation and the gap size between two given scaffolds. The scaffolding structure is represented by a directed graph *S*. The vertices of *S* are oriented scaffolds. Two vertices are connected by an edge if there is a winning bundle between them. The initial scaffolding graph consists of assembly contigs.

The scaffolding algorithm is an iterative process in which scaffolds are incrementally merged and fixed. The merging process uses two separate heuristic methods. In the first one, we merge unambiguously closest neighboring scaffolds, determined by the graph topology and the implied gaps. In the second one, we merge the most strongly connected pairs of scaffolds, determined by bundle weights. In the fixing stage, we tag scaffold regions having low physical coverage and break them apart.

Reference Sequences. For mouse, we used the NCBI build 37 reference sequence, which is from mouse strain C57BL/6J, as are all of the data for the mouse assemblies. We removed chromosome (chr) Y and the "random" records. For human, with one exception (explained below), we used a reference based on the Genome Reference Consortium (GRC) build. We removed the random records. For uniformity, we also removed chr Y, although some of the sequence came from males. We added the mitochondrial genome. There was an exception: For the analysis of contig accuracy for human assembly 1, we used both the GRC reference (as above) and the maternal reference sequence from http://alleleseq.gersteinlab.org, described in Rozowsky et al. ("Coordination between allele-specific expression and binding in a network framework") (under review), based on data from the 1000 Genomes Pilot Project (6), aligned to NCBI human build 36. We further adapted this sequence by adding homozygous SNPs found in DePristo et al., "A framework for variation discovery and genotyping using next generation DNA sequencing data" (under review). We note that the datasets used to define the NA12878 reference are disjoint from the dataset of this work.

There was one other exception. For the analysis of segmental duplications, we used whatever reference sequences had been used to create the segmental duplication databases: the NCBI build 36 sequences for human and mouse.

Description of How Contigs Were Aligned to the Reference. We aligned the contigs in such a way that a given contig could be broken into more than one piece (reflecting a misassembly or bona fide difference with the reference), so that a given contig (or contig piece) could go to only one location and so that alignments of contigs (or contig pieces) having an error rate too high to be biologically correct were discarded. In more detail, we used the program QueryLookupTable (ref. 7, also distributed with ALLPATHS-LG), with arguments K = 12, MM = 12, MC = 0.15, MF = 500:5000:500, and ALIGN_UNALIGNED_BITS = True. With these arguments the code would break contigs into up to three separate pieces in different positions on the reference. Only the best alignment of a given contig (or contig piece) was used, or, in the case of a tie, one was chosen at random. Alignments were scored according to their "reciprocal match rate," defined to be the reciprocal of the weighted mean of the perfect match lengths in the alignment, more precisely $\Sigma(v_i + 1)/\Sigma(v_i + 1)^2$, where v_i are the lengths of the perfectly matching segments or 1, in case of mismatches. Alignments having reciprocal match rate >0.5% were discarded. In rough terms, this procedure had the effect of discarding alignments having more than one mismatch per 200 bases, which were unlikely to be genomically correct. However, with this procedure, large numbers of errors (for example, arising from assembly defects) would not necessarily cause an alignment to be rejected, provided that enough of the alignment was in long perfect matches.

The Two YH Assemblies. The publication of the SOAP YH assembly in ref. 8 refers to the availability of the assembly in two locations. We observed that the assemblies in these two locations are in fact different:

i) GenBank ADDF010000000. This assembly is also available at ftp://ftp.ncbi.nih.gov/genbank/genomes/Eukaryotes/vertebrates_mammals/Homo_sapiens/YH1/Primary_Assembly/unplaced_scaffolds/FASTA/unplaced.scaf.fa.gz.

ii) http://yh.genomics.org.cn. We found this assembly at ftp://public.genomics.org.cn/BGI/yanhuang/Genomeassembly/asm.yanh.scafseq.closure.gz.

The contigs from *(ii)* are about three times longer than those from *(i)*. We therefore used *(ii)* in our analyses.

Treatment of Assembly Ambiguities in Assembly Evaluation. To understand the extent of ambiguity within a given assembly, we count the number of "ambiguous bases," which we define to be the sum over all choices $\{x_1, \ldots, x_n\}$ of the maximum of the lengths of the x_i. Thus the example

... ATC{A, T}GGTTTTTTT{, T, TT}ACAC ...

would be counted as having three ambiguous bases. For a known genome, we can also score the accuracy of a given part of an assembly in alignment to the reference. To do so, for each choice we pick the sequence that best matches the reference.

Known Heterozygous Indels in NA12878. We start from the list of events at ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/pilot_data/release/2010_07/low_coverage/sv/CEU.low_coverage.2010_06.deletions.genotypes.vcf.gz. We selected those events that are asserted to be heterozygous in NA12878, which were not tagged as IMPRECISE, and which were assigned a Phred quality score of at least 40.

Application of SOAPdenovo and ABySS to the Mouse Dataset. ***Data preparation.*** We started from 47 pairs of input files read1.fastq, read2.fastq. EcoP15I reads were trimmed to include only the first 26 bases and the Fosmid reads were trimmed to remove the first 4 bases.

SOAPdenovo. The 47 read1 and 47 read2 fastq files were merged into 4 read1 and 4 read2 fastq files on the basis of the four library types. The SOAP authors provided a perl script (duplication.pl) to remove duplicate paired reads from each library type:

```
duplication.pl ShearedJumps.read1.fastq ShearedJumps.read2.fastq ShearedJumps.read1.fastq.dedup ShearedJumps.read2.fastq.dedup ShearedJumps.dedup_stats
```

Next read correction was done on the fragment pair reads in two steps via the modules KmerFreq and Corrector. The SOAP authors provided new error correction code along with the exact parameters to use:

```
kmer_freq_pfile reads.in -k 17 -f 1 -t 48 -p mouse
correct_error_pread mouse.freq.gz reads.in -k 17 -x 8 -r 50 -l 10 -c 5 -p mouse -f 1 -j 1 -t 48
```

The configuration file required by SOAPdenovo was constructed. One entry in the configuration file is average insert size. Because the range of insert sizes in the jumping libraries is from 1.8 to 2.8 kb, the formerly merged fastq files (before deduplication) were split back out into separate read1 and read2 fastq files on the basis of insert sizes. Average insert size for each library was set (using avg_ins=) in the configuration file. Orientation of read pairs was set in the configuration file using reverse_seq = 0 for the fragment and Fosmid libraries and reverse_seq = 1 for the jumping libraries. Fragment pairs were used for both the contig assembly and scaffold assembly (asm_flags = 3) whereas the jumping and Fosmid libraries were used for the scaffold assembly only (asm_flags = 2). Finally, order of libraries was set from smallest insert size first (fragment pairs rank = 1) to largest insert size last (Fosmid rank = 5). EcoP15I reads were excluded from the SOAPdenovo assembly because their length (26 bases) was shorter than the suggested K-mer length (61) to use in the assembly. The SOAP authors provided an in-house SOAPdenovo version (SOAPdenovo63mer-v1.05) capable of supporting a K-mer length up to 63. A K length of 61 was suggested along with exact parameters to use at each of the four SOAPdenovo assembly stages:

```
SOAPdenovo63mer-v1.05 pregraph -s config.file -K 61 -o 20100913 -a 300 -p 16 –R
SOAPdenovo63mer-v1.05 contig -g 20100913 -M 2 –R
SOAPdenovo63mer-v1.05 map -p 16 -s config.file -g 20100913
SOAPdenovo63mer-v1.05 scaff -F -g 20100913
```

The resulting SOAPdenovo scaffolds were input into the GapCloser program with the aim of closing gaps that remained after the scaffolding stage. The GapCloser program failed to complete in time for manuscript release (total run time is predicted to be ~6 wk), so the SOAP assembly is based on the final output of SOAPdenovo. Total wall clock time from deduplication through SOAPdenovo was 64.2 h.

ABySS. abyss-pe (version 1.2.1) was compiled, enabling max-k of 48 and incorporating openmpi-1.4 and Google sparsehash (revision 52) patched with deallocate.diff. abyss-pe was run using a k-mer size (-k) of 48; 16 processors (np = 16); a trim quality threshold (-q) set to 10; sort –T to a provide a large tmp directory space; and minimum number of pairs to scaffold (n =) set to 20 for fragment pairs, 100 for jumps, and 10 for Fosmids. Total wall clock time for assembly was 105 h.

1. Kim UJ, Shizuya H, de Jong PJ, Birren B, Simon MI (1992) Stable propagation of cosmid sized human DNA inserts in an F factor based vector. *Nucleic Acids Res* 20:1083–1085.

2. Maccallum I, et al. (2009) ALLPATHS 2: Small genomes assembled accurately and with high continuity from short paired reads. *Genome Biol* 10:R103.

3. Butler J, et al. (2008) ALLPATHS: De novo assembly of whole-genome shotgun microreads. *Genome Res* 18:810–820.
4. Pevzner PA, Tang H, Waterman MS (2001) An Eulerian path approach to DNA fragment assembly. *Proc Natl Acad Sci USA* 98:9748–9753.
5. Zerbino DR, Birney E (2008) Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Res* 18:821–829.
6. The 1000 Genomes Project Consortium (2010) A map of human genome variation from population-scale sequencing. *Nature* 467:1061–1073.
7. Brockman W, et al. (2008) Quality scores and SNP detection in sequencing-by-synthesis systems. *Genome Res* 18:763–770.
8. Li R, et al. (2010) De novo assembly of human genomes with massively parallel short read sequencing. *Genome Res* 20:265–272.

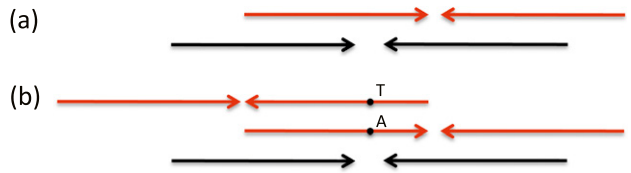

Fig. S1. The ALLPATHS-LG process of fragment pair filling. (*A*) The algorithm tries to close the black pair. It finds another pair (red) that perfectly overlaps the black pair and closes its gap. Sequence from the red pair is inserted into the gap in the black pair, thus closing it. (*B*) Again the algorithm tries to close the black pair, but this time there is a SNP (A or T) between its gap. Two red pairs both overlap the black pair perfectly, providing two separate solutions to its closure, both of which are retained.

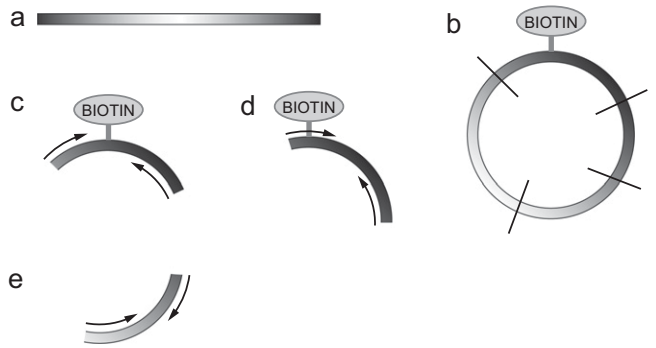

Fig. S2. Artifacts associated with sheared jumping libraries, following the Illumina protocol (1). (*A*) DNA is sheared and size selected, yielding linear fragments. (*B*) The ends of these fragments are biotinylated and then the fragments are circularized and sheared. Fragments of the circles are then enriched for those containing biotin. The ideal fragment is shown in *C*. Two reads enter from opposite sides but do not read the junction. In *D*, one of the reads passes through the junction point, creating a "chimeric" read. In *E*, the ends of a fragment that do not contain a junction point are read, yielding a read pair in opposite orientation to that of *C* and whose true separation on the genome is small.

1. Bentley DR, et al. (2008) Accurate whole human genome sequencing using reversible terminator chemistry. *Nature* 456:53–59.

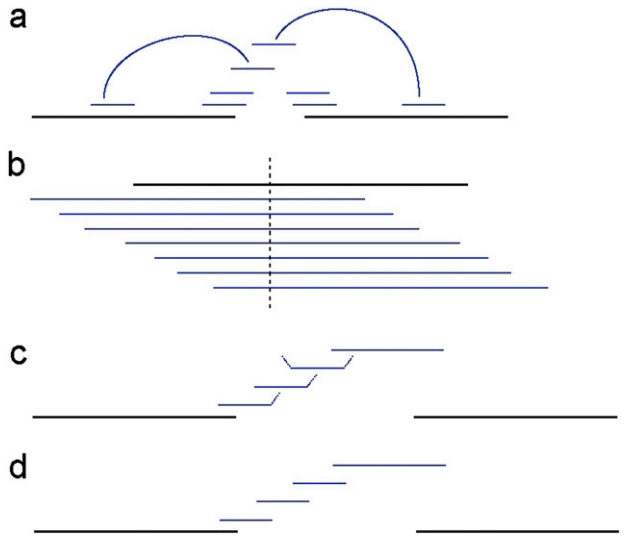

Fig. S3. Gap patching. See *SI Materials and Methods* for *ALLPATHS-LG Algorithms*, *Gap patching*. (*A*) Steps *i–iv* define a pool of oriented reads that might land in a given gap. (*B*) Steps *v* and *vi* define a stack of reads that align to a given read (top); the dotted line shows a column of the stack that "votes" to determine if the corresponding base on the given read is to be changed. (*C*) Step *vii*: All 16-mers that could be party to a bridge across the gap are found; dotted portions of reads are 16-mers that are excluded and then trimmed off the reads. (*D*) Steps *viii–x*: Closures of the gap are found by walking across the gap using perfect overlaps between the reads.

Table S1. Experimental data for human and mouse assemblies (expanded from Table 2)

Species	Library	Size, bp		Read length	Sequence coverage, ×					Physical coverage, ×
		Mean	SD		All	PF	Aligned	Unique	Valid	
Human	Fragment	155	26	101	51.9	41.8	38.4	37.9	36.5	27.8
	Jumping 1	2,283	221	101	21.2	19.0	15.7	14.8	9.2	103.4
	Jumping 2	2,803	271	101	24.7	21.7	18.0	16.9	10.5	146.0
	Fosmid 1 (ShARC)	35,295	2,703	76	0.9	0.6	0.5	0.1	0.1	20.0
	Fosmid 2 (ShARC)	35,318	2,759	101	4.4	3.4	2.5	0.3	0.2	29.5
	Total				103.1	86.5	75.1	70.0	56.5	326.7
Mouse	Fragment	168	32	101	58.6	53.1	49.6	46.6	45.3	37.6
	Jumping 1	1,834	182	101	20.0	17.4	14.7	13.6	8.1	73.2
	Jumping 2	2,289	212	101	17.6	13.9	12.2	10.8	6.9	78.2
	Jumping 3	2,794	220	101	10.4	9.4	8.2	7.6	4.9	67.7
	Long jumping 1	8,611	629	26	2.1	1.7	1.7	1.1	0.6	106.4
	Long jumping 2	6,187	381	26	3.7	1.7	1.7	1.0	0.5	56.4
	Long jumping 3	6,185	347	26	2.8	2.1	2.1	1.3	0.7	78.4
	Long jumping 4	7,030	337	26	1.9	1.7	1.6	1.2	0.7	94.9
	Long jumping 5	10,009	721	26	3.0	2.1	2.1	0.9	0.4	72.2
	Fosmid (Fosill)	38,453	3,557	76	1.4	1.1	1.1	0.1	0.1	23.1
	Total				121.5	104.2	95.0	84.2	68.2	688.1

Some of the information in Table 1 is expanded. To assess the properties of the data, independently of assembly, reads were aligned to the references using Maq (1), after trimming as follows: Jumping library reads were trimmed to 36 bases, but counted as full length for purposes of computing coverage; long jumping (EcoP15I) reads were trimmed to 26 bases; Fosmid (Fosill) reads were trimmed to remove 4 bases from the left; and Fosmid (ShARC) reads were trimmed to remove 28 bases from the left. Library: See Table 1. Size: Mean and SD of observed fragment size distribution. Read length, Sequence coverage, and Physical coverage: see Table 1.

1. Li H, Ruan J, Durbin R (2008) Mapping short DNA sequencing reads and calling variants using mapping quality scores. *Genome Res* 18:1851–1858.

Table S2. Illumina and SRR identifiers for sequence generated

Species	Library type	Illumina lane identifier	SRR run accession no.
Human	Fragment	202PBABXX.1	SRR067787
		202PBABXX.2	SRR067789
		202PBABXX.3	SRR067780
		202PBABXX.4	SRR067791
		202PBABXX.5	SRR067793
		202PBABXX.6	SRR067784
		202PBABXX.7	SRR067785
		202PBABXX.8	SRR067792
		61PHDAAXX.6	SRR067577
		61PHDAAXX.7	SRR067579
		61PHDAAXX.8	SRR067578
	Jumping 1	2025JABXX.1	SRR067771
		2025JABXX.2	SRR067777
		2025JABXX.3	SRR067781
		2025JABXX.4	SRR067776
	Jumping 2	2025JABXX.5	SRR067773
		2025JABXX.6	SRR067779
		2025JABXX.7	SRR067778
		2025JABXX.8	SRR067786
	Fosmid 1	613YAAAXX.1	SRR068214
		613YAAAXX.2	SRR068211
	Fosmid 2	2025TABXX.6	SRR068335
Mouse	Fragment	6141AAAXX.1	SRR067634
		6141AAAXX.2	SRR067650
		6141AAAXX.3	SRR067648
		6141AAAXX.4	SRR067622
		6141AAAXX.5	SRR067649
		6141AAAXX.6	SRR067641
		6141AAAXX.7	SRR067670
		6141AAAXX.8	SRR067636
		613F0AAXX.1	SRR067612
		613F0AAXX.2	SRR067616
		613F0AAXX.3	SRR067615
		613F0AAXX.4	SRR067623
		613F0AAXX.5	SRR067605
		613F0AAXX.6	SRR067646
		613F0AAXX.7	SRR067633
		613F0AAXX.8	SRR067620
		613F1AAXX.1	SRR067606
		613F1AAXX.2	SRR067611
		613F1AAXX.3	SRR067625
		613F1AAXX.4	SRR067601
		613F1AAXX.5	SRR067624
		613F1AAXX.6	SRR067645
		613F1AAXX.7	SRR067635
		613F1AAXX.8	SRR067653
	Jumping 1	61NDRAAXX.1	SRR067607
		61NDRAAXX.2	SRR067669
		61NDRAAXX.3	SRR067603
		61NDRAAXX.4	SRR067660
		61NDRAAXX.5	SRR067619
		61NDRAAXX.6	SRR067658
		61NDRAAXX.7	SRR067604
	Jumping 2	61NGGAAXX.2	SRR067631
		61NGGAAXX.3	SRR067639
		61NGGAAXX.4	SRR067657
		61NGGAAXX.5	SRR067610
		61NGGAAXX.6	SRR067654
	Jumping 3	61NCCAAXX.1	SRR067644
		61NCCAAXX.2	SRR067618
		61NCCAAXX.3	SRR067652
		61NCCAAXX.4	SRR067663

Table S2. Cont.

Species	Library type	Illumina lane identifier	SRR run accession no.
	Long jumping 1	207HCABXX.1	SRR067823
	Long jumping 2	207HCABXX.2	SRR067839
	Long jumping 3	207HCABXX.3	SRR067846
	Long jumping 4	207HCABXX.5	SRR067858
	Long jumping 5	207HCABXX.8	SRR067830
	Fosmid	61NBYAAXX.1	SRR067609

This table is synchronous with Table S1. The third column defines the lanes that were sequenced using the notation flowcell.lane to denote particular lanes from a flowcell. Flowcells suffixed AAXX were sequenced on the GAII whereas those suffixed ABXX were sequenced on the HiSeq. The fourth column gives the Sequence Read Archive (www.ncbi.nlm.nih.gov/sra) SRR accession number for each lane.

Table S3. Repeat content of gaps in ALLPATHS-LG assemblies

Genome	Category	% of genome	% covered	% of gap bases	Enrichment in gaps
Human	LINE	22.1	85.4	36.1	1.6
	LTR retrotransposon	9.2	93.8	6.4	0.7
	SINE	13.8	83.2	25.8	1.9
	Simple repeats	0.9	80.8	2.0	2.2
	DNA transposon	3.5	95.6	1.7	0.5
Mouse	LINE	20.4	66.7	59.9	2.9
	LTR retrotransposon	10.8	82.7	16.5	1.5
	SINE	7.9	95.5	3.1	0.4
	Simple repeat	2.5	87.4	2.8	1.1
	DNA transposon	1.1	97.9	0.2	0.2

Repeat content of ALLPATHS-LG gaps is shown, based on the RepeatMasker track from http://genome.ucsc.edu/cgi-bin/hgTables. Category: repeat class, excluding those occupying <1% of both genomes, including long interspersed elements (LINE), short interspersed elements (SINE), and long terminal repeat (LTR) retrotransposons (1). % of genome: Fraction of genome in the given category. % covered: Fraction of bases in the given category that are covered by the assembly. % of gap bases: Fraction of gap bases that are in the given category. Enrichment in gaps: (gap bases in category)/(genome bases in category), divided by (all gap bases)/(all genome bases).

1. Lander ES, et al., International Human Genome Sequencing Consortium (2001) Initial sequencing and analysis of the human genome. *Nature* 409:860–921.

Table S4. N50 contig size (kilobases) for mouse chr1 assemblies with reduced coverage

		Fraction of jump coverage used		
		50%	75%	100%
Fraction of fragment coverage used	50%	18	19	19
	75%	22	23	23
	100%	24	25	26

Mouse reads (as described in Table 2) were selected on the basis of alignment to chromosome 1 and then randomly subsampled to 100% (full sample), 75%, or 50% of the total. This operation was carried out separately for fragment pairs and jump pairs (including long jumps but not Fosmids), yielding nine assemblies. (Fosmids were included in all assemblies at full coverage.) The N50 contig size for these assemblies is shown. The same analysis for scaffolds is not shown because the N50 scaffold size is ~10 Mb, and because the number of scaffolds in that size range is small, the N50 is not statistically meaningful.

Table S5. Genomic coverage (in percent) for mouse chr1 assemblies with reduced coverage

		Fraction of jump coverage used		
		50%	75%	100%
Fraction of fragment coverage used	50%	92.1	92.5	92.7
	75%	92.8	93.2	93.4
	100%	92.8	93.3	93.5

See Table S4 for comparison as Tables S4 and S5 are parallel.