Appendix

Sizing Bloom filters

We used the tool ntCard to estimate the cardinality F_0 of the set of k-mers and the number of singletons f_1. These counts are used for optimizing both runtime and memory use by allowing us to minimize the size of Bloom filters m and the number of hash functions h used for both B_1 and B_2, the largest filters used. B_1 and B_2 share these parameters (and the same set of hash functions) to allow insertions of each s into B_2 for which $B_1(s) = 1$ without recalculating h hash values. We use the fact that elements inserted into B_2 are either non-singletons or false positives due to B_1. Thus, the expected number of elements n_2 in B_2, is bound by their sum, i.e.,

$$n_2 \leq (F_0 - f_1) + f_1 p_1 \tag{1}$$

where p_1 is the false positive rate of B_1. We note that since p is the *effective* false positive rate after all elements are inserted into B_1, this bound holds strictly and may be overly pessimistic regarding the number of false positives inserted into B_2, however it provides a simple means of setting parameters. To do so, we first recall that B_1 is discarded after loading, while B_2 is maintained and thus its false positive rate p_2 is the rate that affects all downstream queries. A default false positive rate of of $p_2 = 0.01$ is used to work backwards to derive a higher rate p_1, and Bloom filter parameters for both filters were set based on this derived value, using knowledge of F_0 and f_1. To derive p_1, we paired the expressions for the expected false positive rates with the expression for the optimal number of hash functions for a given false positive rate (Mitzenmacher (2002)):

$$p_1 = (1 - e^{\frac{F_0 h}{m}})^h \tag{2}$$

$$p_2 = (1 - e^{\frac{n_2 h}{m}})^h \tag{3}$$

$$h = \frac{m \ln(2)}{F_0}. \tag{4}$$

By plugging the value of h from equation 4 into equation 2, we arrive at $m = \frac{-F \ln(p_1)}{\ln(2)^2}$. Combining this and the above expressions, we arrive at

$$0 = -\ln(2) ln(p_2) - \ln(p_1) \ln(1 - 2^{-\frac{F_0 + (1-p_1) f_1}{F_0}}) \tag{5}$$

for which root-finding methods can be applied to finally extract p_1, the sole remaining unknown.

Currently, we have not yet found similar means of optimizing the sizes of filters B_3 and B_4, as it is unclear how to estimate the number of elements that will be inserted into them in advance. We therefore define their sizes based on empirical observations. For diverse metagenomes, where the number of singletons f_1 may be very close to the cardinality F_0, we expect there to be few junctions, as a junction k-mer must by definition occur at least twice in the data. Based on this observation, we set the expected number of elements in both B_3 and B_4 to be $\frac{F_0}{10}$ and found that this bound was not exceeded on tested datasets. For higher coverage data, where a significantly larger proportion of junctions is expected relative to F_0, we set the size of both filters to be $\frac{F_0}{2}$.

Solid junction counts

Total junction counts listed in the Table 1 below include real junctions, those due to false positives, and dummy junctions inside long linear stretches. We posit that the SYN 64 data set included many more fake (false positive and dummy) junctions as a result of having a much larger proportion of linear stretches, as reflected in the much larger genome fraction and N50 size (relative to HMP) output by Faucet.

	HMP	SYN 64
Total junctions (M)	7.11	9.23
Real junctions (M)	4.55	1.34
genome fraction (%)	27.9	82.3
N50	2290	16707

Table 1.

Inserting into B_4

When inserting into B_4, both the distance and relative orientation between paired-end mates is unknown. Therefore, a tiling scheme such as that seen in Figure 3 cannot be applied. Instead, we seek to ensure that in most cases when querying approximately one insert size away from a given junction u, there will be another junction v such that an extension of u will be paired with an extension of v in B_4. To achieve this end, and to avoid long run times due to pair insertions, we apply the following logic: for each junction u on the first mate, we only insert extensions of a new pair (u, v) if u has no pair in B_4. When a new pair must be inserted, v is chosen to be the first junction found on the second mate. During the insertion process, this logic allows us to break the querying process whenever one previously inserted pair is encountered, and lets us avoid inserting too many pairs into B_4, and thus risking increasing B_4's effective false positive rate.

Additional disentanglement

Other forms of disentanglement include resolution of loops and disentanglement by coverage. Loops are encountered when, e.g., s_a and s_c in Figure 2 are the same unitig, and disentanglement requires unwinding the loop and duplicating the s's sequence to arrive at the walk $[s_b, s, s_c, s, s_d]$. Disentanglement by coverage is allowed only when s is deemed too long for there to be support by junction pairs flanking opposite ends of s, and is applied when the coverage distributions of Contig pairs supporting a certain orientation (e.g., s_a paired with s_c and s_b with s_d for the case presented in Figure 2 is significantly similar, as determined by Two One Sided Tests (Walker and Nowacki (2011)) for each pair. To smooth coverage levels when this test is applied, coverage values are updated each time a cleaning step such as bulge removal is applied. For example, if a bubble includes one low coverage Contig s_1 and one high coverage Contig s_2, as extensions flanked by the same ContigNodes j_L and j_R, and s_2's coverage is sufficiently higher than s_1's, Contig s_1 will be removed, and its average coverage will be assigned to all (expired or fake) junctions on Contig s_2.

Tools comparison details

Tools and flags:
Faucet was run with k= 31
MetaSPAdes 3.9.0, default parameters
Megahit 1.1.1, default parameters
Minia 3 Beta, git commit 4b0a83a, k = 31
LightAssembler, no version information available, downloaded 1/17 from GitHub k = 31
MetaQUAST, 4.4.0, –fragmented flag

Data Sets:
SYN 64 (SRA accession SRX200676), 109M 100 bp paired end mates, I.S. 206
HMP (SRX024329), 149.6 M 100 bp paired end mates, I.S. 213

References

Mitzenmacher, M. (2002). Compressed Bloom filters. *IEEE/ACM Transactions on Networking*, **10**(5), 604–612.

Walker, E. and Nowacki, A. S. (2011). Understanding equivalence and noninferiority testing. *Journal of general internal medicine*, **26**(2), 192–6.