

Supplementary Notes

Supplementary Note 1: Distance metric for clustering algorithm

Assume there are 2 Poisson distributions P_1 and P_2 with density functions

$$p_1(x, \lambda_1) = \frac{e^{-\lambda_1} \lambda_1^x}{\Gamma(x+1)}$$

and

$$p_2(x, \lambda_2) = \frac{e^{-\lambda_2} \lambda_2^x}{\Gamma(x+1)}$$

The Kullback-Leibner divergence from P_2 to P_1 is defined as:

$$D_{KL}(P_1||P_2) = \int_{-\infty}^{\infty} p_1(x) \log \frac{p_1(x)}{p_2(x)} dx$$

or in other words, it is the expectation of the logarithmic difference between the probabilities P_1 and P_2 , where the expectation is taken with regard to P_1 .

The log ratio of the density functions is

$$\log \frac{p_1(x)}{p_2(x)} = x \log \frac{\lambda_1}{\lambda_2} + \lambda_2 - \lambda_1$$

take expectation of this expression with regard to P_1 with mean λ_1 we have

$$D_{KL}(P_1||P_2) = \lambda_1 \log \frac{\lambda_1}{\lambda_2} + \lambda_2 - \lambda_1$$

The metric we used is the distance defined as

$$D(P_1, P_2) = \frac{D_{KL}(P_1||P_2) + D_{KL}(P_2||P_1)}{2} = \frac{1}{2}(\lambda_1 - \lambda_2)(\log \lambda_1 - \log \lambda_2)$$

Supplementary Note 2: Coverage re-estimation

The re-estimation is basically carried out by following two steps.

1. From nodes coverage, estimate edges' value by quadratic unconstrained optimization of the least-square function:

$$\frac{1}{2} \sum_i l_i ((\sum e_i^+ - c_i)^2 + (\sum e_i^- - c_i)^2)$$

where l_i and c_i is the length and coverage of a node i in the graph;

$\sum e_i^+$ and $\sum e_i^-$ indicates sum of the values of incoming and outgoing edges from i respectively. The above function and be rewritten as:

$$f(x) = \frac{1}{2} x^T Q x + b^T x + r$$

and then being minimized by using gradient-based method.

2. Update nodes' coverage based on itself and its neighbor edges' measures.

The calibration is iterative until no further improvements are made or a threshold loop count is reached.

Supplementary Note 3: Path finding algorithm

Algorithm 1 demonstrates the path finding module in general. In which, function

`shortestTree(vertex, distance) : (V, Z) → Vn`

from line 3 of the algorithm's pseudo code builds a shortest tree rooted from \vec{v} , following its direction until a distance of approximately d (with a tolerance regarding nanopore read error rate) is reached. This task is implemented based on Dijkstra algorithm. This tree is used on line 4 and in function `includedIn()` on line 19 to filter out any node or edge with ending nodes that do not belong to the tree.

Basically, the algorithm keeps track of a stack that contains sets of candidate edges to discover. During the traversal, a variable d is updated as an estimation for the distance to the target. A hit is reported if the target node is reached with a reasonable distance *i.e.* close to zero, within a given tolerance (line 21). A threshold for the traversing depth is set (150) to ignore too complicated and time-consuming path searching.

Algorithm 1: Pseudo-code for finding paths connecting a bridge with 2 ends.

Data: Assembly graph $G\{V, E\}$
Input: Bridge $B = (\vec{v}_1, \vec{v}_2)$ with two ending unique bidirected nodes \vec{v}_1, \vec{v}_2
Output: Set of candidate paths P connecting B

```

1 begin
2    $d := B.length()$                                 // length of the bridge or the distance between 2 ending nodes
3    $M := shortestTree(\vec{v}_2, d)$                         // build shortest tree from  $\vec{v}_2$  with range  $d$ 
4   if  $M.contains(\vec{v}_1)$  then
5      $S := new Stack()$                                 // stack of sets of edges to traverse
6      $edgesSet := getEdges(\vec{v}_1)$                       // get all bidirected edges going from  $\vec{v}_1$ 
7      $S.push(edgesSet)$ 
8      $p := new Path(\vec{v}_1)$                             // init a path that has  $\vec{v}_1$  as root
9     while true do
10       $edgesSet := S.peek()$ 
11      if  $edgesSet.isEmpty()$  then
12        if  $p.size() \leq 1$  then
13          break                                         // stop the loop when there is no more edge to discover
14           $S.pop()$ 
15           $d += p.peekNode.length() + p.popEdge().length()$ 
16        else
17           $curEdge := edgesSet.remove()$ 
18           $\vec{v} := curEdge.getOpposite(p.peekNode())$ 
19           $S.push(getEdges(\vec{v}).includedIn(M))$ 
20           $p.add(curEdge)$ 
21          if reach  $\vec{v}_2$  with reasonable  $d$  then
22             $P.add(p)$ 
23             $d -= \vec{v}.length() + curEdge.length()$ 
24    return  $P$ 

```

It is worth to mention that the *length()* functions for node and edge are totally different. While the former returns the length of the sequence represented by the node, *i.e.* contig from short-read assembly, the latter is usually negative because an edge models a link between two nodes, which is normally an overlap (except for composite edges). For example, in a *k-mers* SPAdes assembly graph, the value of an edge is $-k + 1$.

On the other hand, if external binning algorithm is employed, the resulting output must be converted into a text file that specifies the corresponding bin of every contigs. By that, each line of the file would be: `< contig_ID > < bin_ID >` where `bin_ID = 0` indicates unspecified binned contigs.

Supplementary Figures and Tables

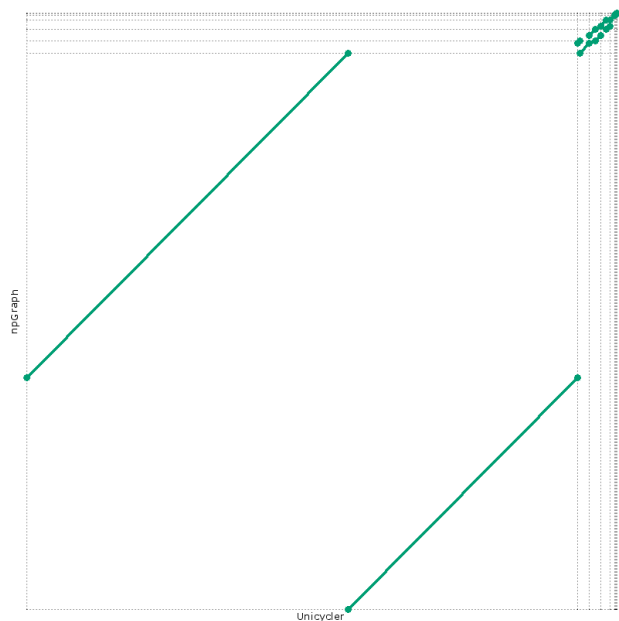
Supplementary Table 1: Benchmarking npGraph against npScarf versions, hybridSPAdes and Unicycler hybrid assembler with the synthetic data set.

Method	Assembly size (bp)	#Contigs	N50 (bp)	Mis-assemblies	Mismatch (per 100Kbp)	Indel (per 100Kbp)
random sequences no repeats						
npScarf	4110000	3	4000000	0	0.00	0.00
npScarf_wag	4109516	3	4000000	0	0.00	0.00
npGraph-bwa	4110000	3	4000000	0	0.00	0.00
npGraph-mm2	4110000	3	4000000	0	0.00	0.00
hybridSPAdes	4110231	3	4000077	0	0.00	0.07
Unicycler	4110000	3	4000000	0	0.00	0.00
random sequences some repeats						
npScarf	4110940	3	4002715	0	0.00	0.66
npScarf_wag	4437094	7	2795129	0	0.00	1.07
npGraph-bwa	4110000	3	4000000	0	0.00	0.00
npGraph-mm2	4110000	3	4000000	0	0.00	0.00
hybridSPAdes	4107566	3	3999364	0	0.02	0.02
Unicycler	4110000	3	4000000	0	0.00	0.00
random sequences many repeats						
npScarf	4316934	8	3963485	24	11.55	34.54
npScarf_wag	5215965	16	1515563	37	0.32	7.22
npGraph-bwa	4110000	3	4000000	0	0.32	0.15
npGraph-mm2	4110000	3	4000000	0	0.32	0.15
hybridSPAdes	4108190	3	3999621	0	0.68	0.15
Unicycler	4110000	3	4000000	0	0.32	0.15
<i>Acinetobacter</i> A1						
npScarf	3912299	3	3870269	4	4.74	13.02
npScarf_wag	3945166	3	3906368	1	7.00	14.02
npGraph-bwa	3918374	2	3909643	0	16.34	0.61
npGraph-mm2	3885898	2	3877167	1	18.05	1.03
hybridSPAdes	3929948	53	3353679	0	35.48	3.22
Unicycler	3917745	2	3909014	0	2.50	0.13
<i>Acinetobacter</i> AB30						
npScarf	4512464	7	4304628	35	57.95	72.87
npScarf_wag	5315235	13	1267710	136	73.55	8.15
npGraph-bwa	4335342	2	4148952	1	16.93	1.45
npGraph-mm2	4335790	1	4335790	0	6.97	0.25
hybridSPAdes	4337369	3	2701005	0	12.80	1.39
Unicycler	4333041	1	4333041	1	6.42	0.53
<i>E. coli</i> K12 MG1655						
npScarf	4649902	2	4641702	4	14.94	34.35
npScarf_wag	4687952	3	4641732	0	6.55	1.96
npGraph-bwa	4641743	1	4641743	0	4.50	0.43
npGraph-mm2	4641820	1	4641820	0	3.88	0.26
hybridSPAdes	4644555	1	4641036	0	0.62	0.09
Unicycler	4641650	1	4641650	0	3.43	0.26
<i>E. coli</i> O25b H4-ST131						
npScarf	5245913	3	5095571	7	7.05	18.81
npScarf_wag	5292700	3	3469617	9	9.03	1.55
npGraph-bwa	5237821	7	4049493	1	3.38	0.31
npGraph-mm2	5249799	3	5110117	0	2.40	0.15
hybridSPAdes	5252762	8	4258948	2	5.43	0.57
Unicycler	5249442	3	5109760	0	4.02	0.27

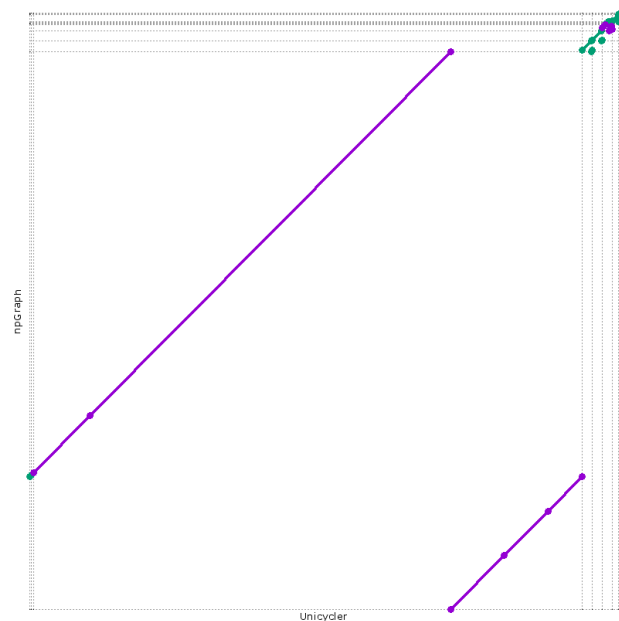
Continued on next page

Supplementary Table 1 – Continued from previous page

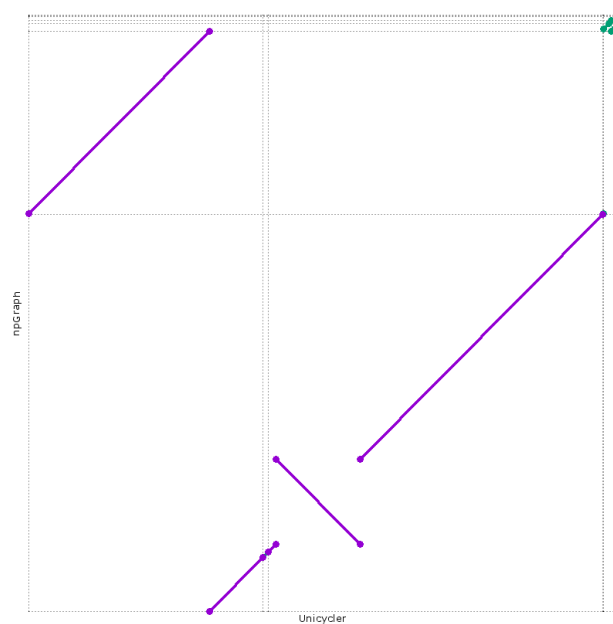
Method	Assembly size (bp)	#Contigs	N50 (bp)	Mis- assemblies	Mismatch (per 100Kbp)	Indel (per 100Kbp)
<i>Klebsiella</i> 30660 NJST258 1						
npScarf	5559772	7	5259053	4	17.18	13.48
npScarf_wag	5613780	7	5268535	6	1.59	1.92
npGraph-bwa	5534843	8	5263229	0	3.15	0.76
npGraph-mm2	5534878	8	5263264	0	2.75	0.74
hybridSPAdes	5545668	8	5545668	2	4.95	0.09
Unicycler	5537860	9	5263196	0	1.34	0.51
<i>Klebsiella</i> MGH 78578						
npScarf	5729304	5	5316429	12	14.90	20.27
npScarf_wag	5754443	5	3026286	16	8.17	2.56
npGraph-bwa	5695801	7	5311745	1	12.65	1.25
npGraph-mm2	5696302	6	5315267	0	6.06	0.44
hybridSPAdes	5706470	11	5315273	1	3.82	0.67
Unicycler	5694231	14	5315096	0	5.38	0.21
<i>Klebsiella</i> NTUH-K2044						
npScarf	5471696	2	5249198	6	4.82	8.55
npScarf_wag	5530559	3	5249369	2	2.25	1.35
npGraph-bwa	5472629	2	5248476	0	2.52	0.31
npGraph-mm2	5472655	2	5248503	0	1.21	0.26
hybridSPAdes	5473572	2	5248894	0	0.44	0.15
Unicycler	5472697	2	5248545	0	2.41	0.35
<i>Mycobacterium tuberculosis</i> H37Rv						
npScarf	4498245	4	4402238	8	5.15	2.68
npScarf_wag	4506056	4	4410942	3	6.81	2.59
npGraph-bwa	4411406	1	4411406	0	1.88	0.43
npGraph-mm2	4411532	1	4411532	0	0.68	0.00
hybridSPAdes	4413942	1	4410519	0	0.75	0.11
Unicycler	4411538	1	4411538	0	2.22	0.34
<i>Saccharomyces cerevisiae</i> S288c						
npScarf	11986800	24	796769	51	62.12	21.46
npScarf_wag	12003203	21	917017	21	69.14	5.47
npGraph-bwa	11921736	40	913090	3	38.04	1.94
npGraph-mm2	11920984	38	913198	2	20.66	0.95
hybridSPAdes	12027533	45	770543	5	32.58	1.94
Unicycler	11847655	72	909114	0	21.81	1.04
<i>Shigella dysenteriae</i> Sd197						
npScarf	4586075	173	36560	55	120.14	111.59
npScarf_wag	5462918	92	98791	105	147.48	79.28
npGraph-bwa	4564058	6	4369264	5	80.64	11.16
npGraph-mm2	4558920	6	4364264	7	75.51	10.98
hybridSPAdes	4519131	23	821249	96	9.57	1.42
Unicycler	4560901	3	4369231	0	11.88	1.05
<i>Shigella sonnei</i> 53G						
npScarf	6441461	20	1953896	82	164.02	219.52
npScarf_wag	-	-	-	-	-	-
npGraph-bwa	5211544	4	4988532	0	14.53	0.31
npGraph-mm2	5211527	4	4988519	0	8.56	0.17
hybridSPAdes	5223875	8	2195455	2	41.92	0.06
Unicycler	5220517	5	4988548	0	7.39	0.52
<i>Streptococcus suis</i> BM407						
npScarf	2183951	3	2146594	0	21.51	9.17
npScarf_wag	2289880	3	1493189	1	3.17	1.96
npGraph-bwa	2154623	6	2131479	1	5.25	0.28
npGraph-mm2	2149876	6	2146774	0	1.44	0.28
hybridSPAdes	2172703	2	2146237	0	6.82	0.09
Unicycler	2170829	2	2146250	0	2.67	0.32



(a) *Citrobacter freundii* CAV1374

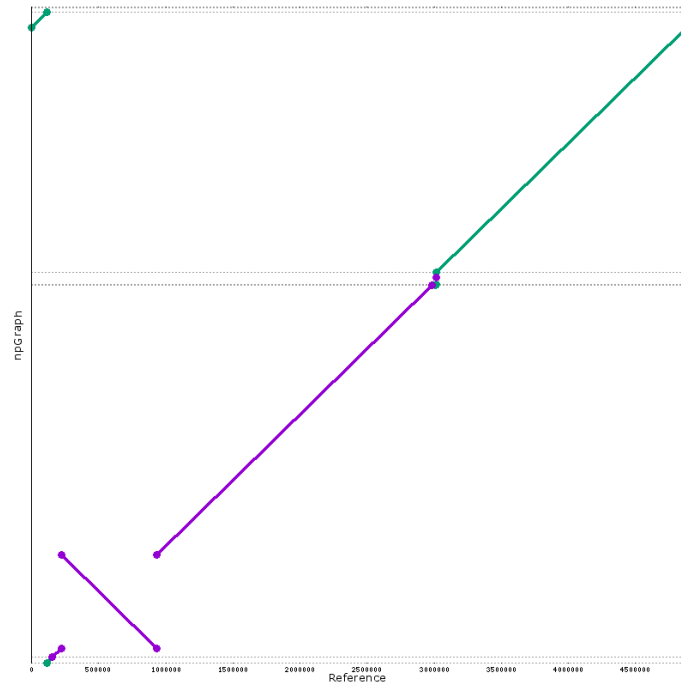


(b) *Klebsiella oxytoca* CAV1015

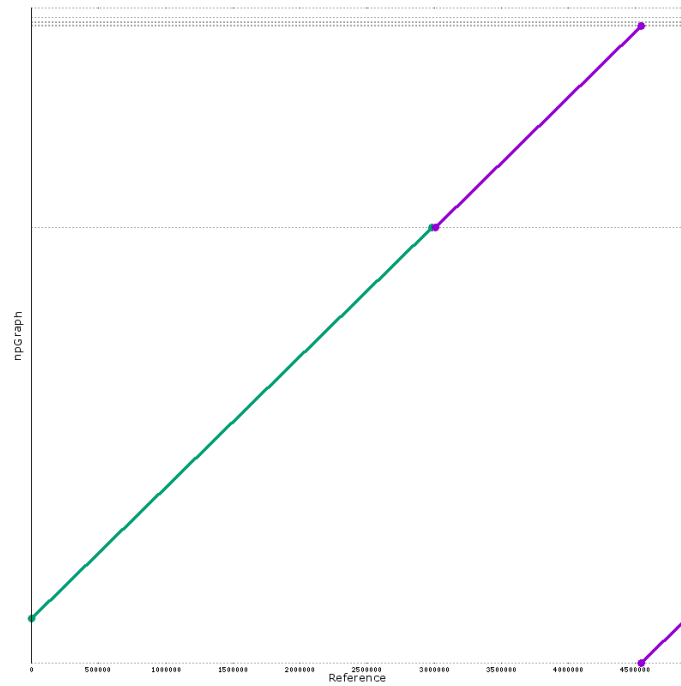


(c) *Enterobacter cloacae* CAV1411

Supplementary Figure 1: Dotplot generated by MUMmer for assembly results of **Unicycler** versus **npGraph**. Structural agreements between two methods were found in (a) *C.frendii* and (b) *K.oxytoca* assembly contigs. On the other hand, for (c) *E.cloacae* sample, there was a disagreement detected between 2 largest contigs given by two assembly algorithms. This case is investigated more thoroughly by using a reference from a same bacterial strain in Figure 2.



(a) *E. cloacae* Unicycler assembly versus reference genome



(b) *E. cloacae* npGraph assembly versus reference genome

Supplementary Figure 2: Alignments of an *Enterobacter cloacae* reference genome to assembly sequences generated by (a) Unicycler and (b) npGraph. While the former presents a structural variant, the latter is virtually an 1-to-1 mapping.