BMC Biology

SOFTWARE

Open Access

ntSynt: multi-genome synteny detection using minimizer graph mappings

Lauren Coombe[1], Parham Kazemi[1], Johnathan Wong[1], Inanc Birol[1,2] and René L. Warren[1*]

Abstract

Background With the growing availability of reference-grade genome assemblies across diverse taxa, there is an increasing need for efficient and scalable tools for multi-species comparative genomics, including synteny detection. Here, we introduce ntSynt, a scalable utility for computing large-scale multi-genome synteny using an alignment-free, minimizer graph-based approach.

Results Through benchmarking on vertebrate genomes (~ 3 Gbp) and 11 bee genomes, we demonstrate that ntSynt produces accurate synteny maps with high genome coverage (79–100%) while using modest computational resources (~ 2 h, 34 GB memory).

Conclusions ntSynt's efficiency and scalability enable large-scale comparative analyses across the tree of life, providing a robust foundation for downstream comparative and functional genomic studies.

Keywords Synteny, Comparative genomics, Multi-genome analysis, Minimizers, Bloom filters

Background

Rapid advances in both genome sequencing technologies and assembly algorithms have led to an explosion of genome sequences assembled at chromosome scale [1]. From pangenomics initiatives, such as the Human Pangenome Project [2], to programs assembling a wide variety of non-model organisms, like the Earth BioGenome Project [3, 4], researchers now have unprecedented access to high-quality reference genomes. Robust bioinformatics tools are crucial to leverage this influx of data for comparative genomics studies, which can enable important insights into genomic influences on phenotypes, genomic diversity, and genome synteny [5–8].

Genome synteny studies center on analyzing the conservation of genome structure within or between species [9–12]. The analysis can focus on large-scale macrosynteny, which tolerates smaller rearrangements within synteny blocks, or more fine-grained microsynteny [13–15]. Macrosynteny is characterized by a high level of chromosome-level conservation among the compared genomes, while microsynteny analyzes conservation of more localized segments [13, 16]. Importantly, analyzing macrosynteny can enable the inference of ancestral linkage groups [17, 18]. Assessing synteny between genomes enables deeper insights into the evolution of genome structure, and supports a deeper functional understanding of genomes [19]. Synteny blocks can be broken at various types of genomic rearrangements, including inversions, translocations, insertions, and deletions [9, 20].

Studies of genomic synteny, such as research by Nadeau and Taylor [21] into the conservation between human and mouse genomes in the 1980s, predates the sequence assembly of large genomes. Many early synteny studies relied on the labor-intensive and low-resolution laboratory technique of chromosome painting

*Correspondence:
René L. Warren
rwarren@bcgsc.ca
[1] Canada's Michael Smith Genome Sciences Centre at BC Cancer, 570 W 7th Ave, Vancouver, BC V5Z 4S6, Canada
[2] Department of Medical Genetics, University of British Columbia, Vancouver, BC V6T 1Z3, Canada

[22], which involves hybridizing chromosome-specific fluorescent probes to cytogenetic slides. Starting in the 2000s, the increased availability of genomic resources lead to the development of multiple anchor-based bioinformatic synteny detection tools, including DAGchainer [19], SynChro [10], MCScanX [23], and DRIMM-Synteny [24]. Each of these tools detect the conserved order of anchors or markers [9, 25], generally homologous genes, to detect synteny, although DRIMM-Synteny can use anchors based on other definitions of similarity. SynChro uses reciprocal best hits from the input genes to construct synteny backbones, and MCScanX chains collinear blocks based on BLASTP [26] outputs using a dynamic programming approach. DAGChainer and DRIMM-Synteny both use graph-based approaches, with DAGChainer traversing a directed acyclic graph, and DRIMM-Synteny using an A-Bruijn graph-based representation.

More recently, improvements in sequence mapping and alignment algorithms have enabled tools to detect conserved stretches of DNA without requiring gene annotations. By using a whole genome approach, as opposed to sparse anchors, they compute synteny between the entire input genome sequences. Satsuma [27], SyMap [15], SyRI [8], and halSynteny [14] compute synteny blocks on pairs of input genomes. Satsuma aligns the input genomes using a fast Fourier transform approach, while the others require sequence alignments as inputs. SyMap uses MUMmer [28] alignments, which are clustered into anchors and chained using dynamic programming. SyRI is designed to compare chromosome-level genome assemblies, and utilizes minimap2 [29] or MUMmer alignments. While Progressive Cactus [30], the aligner recommended for halSynteny, can compute multi-genome sequence alignments, halSynteny itself compares pairs of genomes.

As genome assemblies proliferate across species, genera, and higher taxonomic groups, a tool that enables simultaneous comparison of multiple genomes, rather than pairwise alone, offers significant benefits in analytical scope and computational efficiency. When computing multi-genome synteny blocks, SibeliaZ [31] and SyntenyPortal [11] are two state-of-the-art utilities. SibeliaZ computes multi-genome synteny blocks using compacted de Bruijn graphs, with graph traversal using carrying paths. With this approach, SibeliaZ is suited to computing locally collinear synteny blocks between multiple similar genomes, such as those belonging to different strains of a species, which can lead to lower synteny coverage when comparing more genomes with higher sequence divergence. Finally, SyntenyPortal is a web application that computes synteny blocks between genome assembly builds from the UCSC Genome Browser [32]. SyntenyPortal utilizes pre-computed pairwise alignments from this database with inferCars [33] to determine synteny block coordinates at four pre-specified resolutions. Therefore, none of these tools offer flexible and customizable computation of large-scale synteny blocks on multiple genomes.

Today, an increasing number of bioinformatics tools are utilizing minimizer sketches [34], which represent underlying sequences using a particular subset of *k*-mers (substrings of length *k*) for various applications, including mapping [29], estimating sequence divergence [35], and assembly scaffolding [36, 37]. Sketching greatly reduces the computational cost of comparing and analyzing sequences, making it an attractive approach for scalable tool development in support of large-scale genomics research. In this work, we adapt the minimizer graph introduced in ntJoin [36] for analyzing genome synteny. ntJoin [36] utilizes undirected minimizer graph sketches to map genome sequences to one another for reference-guided scaffolding.

Here, we introduce ntSynt, a scalable utility for computing large-scale and multi-genome synteny blocks. ntSynt uses lightweight, Bloom filter [38] guided minimizer sketches to create an undirected minimizer graph, which is then leveraged for synteny block computation (Fig. 1, Additional file 1: Fig. S1). After graph simplification and breaking synteny blocks at putative large indels, the graph is extended with increasingly dense minimizer sketches to enhance the resolution of the synteny block coordinates. Finally, collinear blocks are merged to output the final synteny blocks. Each step of the ntSynt pipeline is flexible to user input genome sequences (assemblies) and parameters, making ntSynt a broadly useful utility. We show how ntSynt produces contiguous and accurate synteny blocks for genomes of increasing divergences, relatively quickly and with a low memory footprint, enabling an array of comparative genomics study designs.

Implementation

ntSynt takes two or more genomes (reference-grade or draft assembly sequences) in FASTA format as input, maps the input sequences to one another using minimizers, and outputs the synteny blocks in a BED-like format (Additional file 1: Fig. S1, Table S1).

Common Bloom filter construction

First, ntSynt constructs a common, succinct Bloom filter data structure to efficiently capture *k*-mers shared across all genomes, forming the basis for downstream minimizer sketches. In the first step of the pipeline, ntSynt generates a common Bloom filter, which contains the *k*-mers found in all input genomes to be

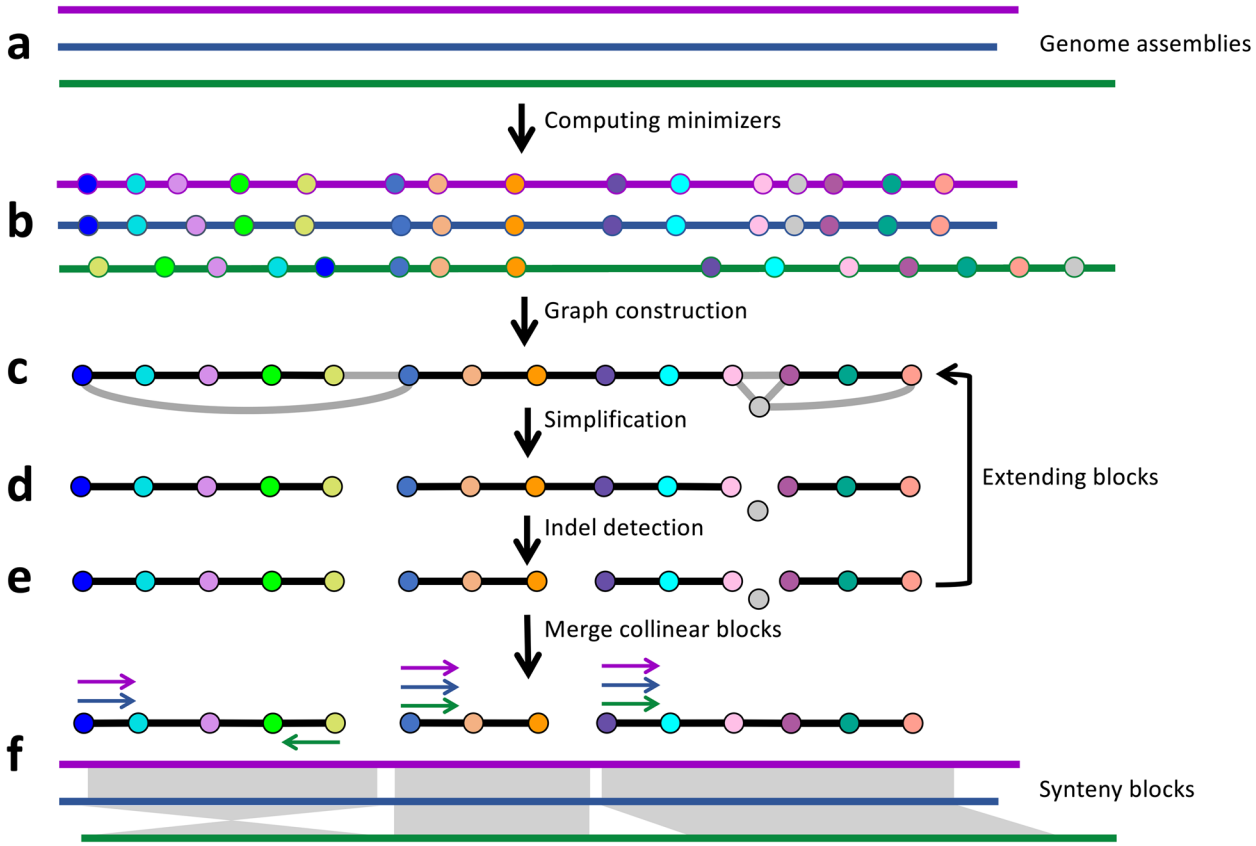

Fig. 1 Schematic of the ntSynt workflow. **a** Multiple genome assemblies are provided as input to ntSynt, represented with purple, blue, and green lines. **b** First, minimizer sketches are computed from each input genome assembly using the *indexlr* functionality of btllib [39] with a Bloom filter comprised of the *k*-mers common to all assemblies. The minimizers are shown as circles, with the same fill color indicating identical minimizers. **c** Next, the minimizers that are found in all assemblies and are unique in each individual assembly are used to create an undirected minimizer graph, where the nodes are minimizers and edges are created between adjacent minimizers. The edge weights correspond to the number of genome assemblies that support the edge (represented as black for full assembly support and grey for partial assembly support, respectively). **d** The graph is simplified and edges are filtered to produce a series of linear graphs, or paths, that correspond to the initial synteny blocks. **e** The paths are broken at putative indels, and the graph can be extended to increase the block resolution using higher density minimizer sketches (arrow to (**c**)). **f** Finally, collinear blocks are merged, and the final synteny blocks are output. The colored arrows indicate the relative orientation of the synteny blocks for each genome assembly. Synteny blocks between genomes are shown using the grey ribbons, with direct synteny represented by rectangles and inverted synteny between the blue and green sequences represented by the crossing ribbon

compared, using a multi-level cascading Bloom filter approach (Additional file 1: Figs. S1 and S2). Bloom filters are data structures which implement set membership using a bit array (array of 1s and 0s), allowing probabilistic detection of element presence and deterministic confirmation of absence [38]. For n input assemblies, there will be n Bloom filters (BF) in the cascade, with at most two initialized and stored in memory at a given time. The size of each BF (*bfsize*, in bits) is determined using the input genome sizes and the specified false positive rate (*--fpr*, default 0.025) as shown in Eq. 1, where *genome_size* is the largest input genome size in base pairs, and *fpr* is the false positive rate.

$$bfsize = \left\lceil -\frac{\text{genome\_size}}{log_e(1-fpr)} \right\rceil \quad (1)$$

To initialize the cascade, the sequences in assembly *1* are *k*-merized (substrings of length *k*) using ntHash2 [40] and inserted into BF level *1*. Then, for each subsequent assembly i ($2 \leq i \leq n$), a new BF level i is initialized. If $i > 2$, the level $i-2$ BF is deallocated prior to the level i BF initialization. The sequences in genome i are *k*-merized, and queried against BF level $i-1$. The *k*-mers that are present in BF level $i-1$ are inserted into BF level *i*. Once all input sequences have been *k*-merized and inserted into the corresponding BF level, the final BF level n, termed the *common Bloom filter*, is then used for computing minimizer sketches in subsequent steps.

Constructing the initial undirected minimizer graph

ntSynt generates a compact representation of all input genomes using shared minimizers and connects them into an initial undirected graph capturing adjacency relationships across genomes. Minimizer sketches are generated for each input sequence using the *indexlr* utility in btllib [39], which generates minimizers based on the approach described in Roberts et al. 2004 [34]. Briefly, for a given sequence, the canonical (reverse-complement invariant) hash values for *w* (window size, *-w,* default 1000) adjacent *k*-mers (substrings of length *k, -k,* default 24) are computed using ntHash2 [40]. Only *k*-mers that are present in the common Bloom filter are considered when selecting the window's minimizer. The *k*-mer with the smallest hash value is chosen as the minimizer for the given window, and the output minimizer hash value is computed using a second hash function. Minimizer sketches for all input genome sequences are generated by sliding the window across all sequences, and repeating the operation for each window, only adding to the sketch when there is a new minimizer. In addition to the minimizer hash value, *indexlr* also reports the position, strand and corresponding *k*-mer sequence for each minimizer.

The initial undirected minimizer graph is constructed as described in ntJoin [36], where the nodes are minimizers, and edges are created between minimizers that are adjacent in at least one genome sequence input. The edge weights correspond to the number of input genomes in which the minimizers are adjacent ($1 \leq \text{edge\_weight} \leq n$, where n is the number of input genomes). Only minimizers that are found in all input genomes and are unique in each individual genome will be retained and added to the minimizer graph.

Filtering the minimizer graph

After generating the initial minimizer graph, ntSynt removes noisy or inconsistently supported minimizers from the graph to simplify paths and retain robust adjacency relationships. The minimizer graph is simplified by removing noisy minimizers that disrupt linear graph paths (Additional file 1: Fig. S3). First, we define fully anchored, partially anchored, and unanchored nodes. Fully anchored nodes have a degree of two and the sum of their incident edge weights is the maximum value ($2n$, where n is the number of input genomes). Partially anchored nodes have a degree of three, and one incident edge weight is n. All other nodes are considered unanchored. For each edge (u, v), where both u and v are partially anchored nodes, if there is an alternate path of length two (u, z, v) between the nodes, the middle node z is marked as noisy and removed from the graph. The weight of edge (u, v) is set as n after this removal. Following graph simplification, all remaining edges with a weight less than n are removed from the graph.

Finding initial synteny blocks

Linear paths through the filtered minimizer graph are converted into preliminary synteny blocks, with orientations and coordinates assigned for each genome. The initial synteny blocks are identified from linear paths through the filtered minimizer graph. A given linear graph path can be converted to synteny block coordinates using the positions of the minimizers for each input sequence in that graph (Additional file 1: Fig. S4). Each linear path is first broken at edges which connect minimizers from different sub-sequences (or contigs) in any of the input genome sequences. Then, the start and end coordinates of the synteny block in each input sequence are computed from the terminal minimizers in the path. The orientation of the block in a given sequence input is determined by assessing if the minimizer positions in the graph are largely (≥ 90%) increasing or decreasing, which then assigns the orientation as "+" (forward) or "−" (reverse), respectively. If a block cannot be oriented, it is not included in the ntSynt output.

Breaking synteny blocks at indels

Edges in the minimizer graph are evaluated for large insertions or deletions, and blocks are broken where variation exceeds a defined threshold. To break synteny blocks at indels, the interarrival distance, defined as the distance between adjacent minimizers, is computed for each edge in the minimizer graph paths (Eq. 2, Additional file 1: Fig. S5). For n input genomes, for each edge (u, v), where a_{pos_x} is the position of minimizer a in genome x, the indel score is computed as:

$$\begin{aligned} \text{interarrival\_max}_{u,v} &= \max(|u_{pos_i} - v_{pos_i}| \textit{for } i \textit{ in } 1..n) \\ \text{interarrival\_min}_{u,v} &= \min(|u_{pos_i} - v_{pos_i}| \textit{for } i \textit{ in } 1..n) \\ \text{indelscore}_{u,v} &= \text{interarrival\_max}_{u,v} - \text{interarrival\_min}_{u,v} \end{aligned} \tag{2}$$

If the indel score for a given edge is greater than the specified indel score threshold (*--indel*), the edge is removed from the path, breaking the synteny block at the large indel.

Dynamic minimizer graph extension with decreasing window size

Synteny blocks are refined by adding additional minimizers computed with smaller window sizes to increase block resolution in uncovered regions. The existing minimizer graph, or paths, can be extended using minimizers computed with decreasing window sizes (Additional file 1: Fig. S6). For a given window size *w_round* less than

the original *w*, minimizers are computed (using the same *k*-mer size and common Bloom filter) on regions of each input genome sequence that are not covered by a synteny block. These sketches are used to dynamically extend the graph in both directions. Then, ntSynt repeats the graph simplification, filtering and indel detection steps to update the synteny blocks. This process of graph extension to increase block resolution using lower window sizes can be performed for any number of rounds (specified using a list of decreasing window sizes, *--w_rounds*).

Merging collinear synteny blocks

Adjacent synteny blocks with consistent orientation and contig assignments are merged to form longer, collinear blocks. The resulting synteny blocks are sorted, filtered based on size (blocks greater than *--block_size* are retained) and collinear synteny blocks separated by up to a threshold number of bases (*--merge*) are merged. Synteny blocks are considered collinear if they have consistent contig IDs, strands and positions (Additional file 1: Fig. S7), and are not separated by an indel.

Final output synteny blocks

The final synteny blocks are output in a BED-like format, where each synteny block is assigned a unique block ID (Additional file 1: Table S1). Between each pair of synteny blocks, the final column specifies the reason for the discontinuity, which includes inconsistent contig IDs, orientations or positions, or exceeding the indel or merge thresholds.

Software implementation

This section describes the software design, dependencies, and configurable parameters of the ntSynt pipeline. The ntSynt pipeline is driven by a Snakemake [41] file, which is launched using a Python wrapper script. The common Bloom filter construction is implemented in C++, and all other algorithms are implemented in Python. ntSynt is available on GitHub (https://github.com/bcgsc/ntSynt), and can be installed using conda. ntSynt requires the user to supply an estimated maximum sequence divergence (*--divergence %*) between the input genomes, which in turn selects presets with default settings for the *--block_size*, *--indel*, *--merge*, and *--w_rounds* parameters (Additional file 1: Table S2). However, all parameters can be configured and independently fine-tuned by the user, and, when specified, override the presets selected by ntSynt in response to the input % divergence estimate.

Evaluation

We first benchmarked ntSynt against simulated genome rearrangements to assess accuracy and contiguity of synteny block detection under varying mutation rates. Four separate rearrangements of the human reference genome (T2T build) were simulated using SURVIVOR [42] v1.0.7, with single-nucleotide variant (SNVs) and indels introduced at various rates using pIRS [43] v2.0.2. SURVIVOR was configured to simulate two translocations between 10 and 50 kbp, 20 inversions between 10 and 50 kbp, and 20 indels between 50 and 100 kbp. For each rearranged genome sequence, SNVs were simulated at 0.1%, 1–5% (step size 1), with indels simulated at 0.01%, 0.1–0.5% (step size 0.1), respectively. For pairwise synteny block detection, one rearranged human genome was compared to the human reference genome using ntSynt (v1.0.0), SibeliaZ [31] (v1.2.5), halSynteny [14] (v2.6.8), and SyRI [8] (v1.6.3). As suggested in their documentation, SibeliaZ ran using the *-n* option to skip the alignment step, then the output was passed to maf2synteny [44] (v1.2) to generate the final synteny blocks. The input pairwise alignments for halSynteny and SyRI were generated using Progressive Cactus [30] (v2.6.8) and minimap2 [29] (v2.17-r941), respectively. The parameters used for each of these runs can be found in Additional file 1: Table S3. For multiple genome synteny block detection, the four rearranged human genomes with various SNV and indel rates and the human reference genome were compared using ntSynt and SibeliaZ (Additional file 1: Table S3). When calculating synteny coverage, we used the average lengths of synteny blocks from each compared genome divided by the length of the smallest genome. The accuracy of each generated synteny block was assessed by comparing each synteny block to the ground truth synteny blocks (Additional file 1: Fig. S8). All runs involving simulated genome sequences were run in triplicate for benchmarking assessments.

Synteny detection was applied to multiple reference and de novo genome assemblies across primates, rodents, and bees to demonstrate scalability and accuracy in real-world datasets. We first compared the human (GRCh38), bonobo (Mhudiblu_PPA_v0), chimpanzee (Clint_PTRv2) and gorilla (Kamilah_GGO_v0) reference genomes using ntSynt and SibeliaZ (Additional File 1: Table S4). Mash [35] (v2.3, *-p* 12 *-s* 10,000) was used to estimate the maximum sequence divergence between these input genomes, and the *--divergence* parameter for ntSynt was subsequently set at 1.7. All other ntSynt parameters were kept at default settings, and SibeliaZ was run using the parameters listed in Additional file 1: Table S3. Synteny blocks between the same reference

genome builds were also obtained using the SyntenyPortal [11] web application (resolution = 150 kbp). We compared the four-genome synteny blocks that were larger than 150 kbp using the gggenomes [45] R package (v0.9.12.9000). To assess a large observed deletion in chromosome 11 of the gorilla genome, we aligned Pacific BioSciences (PacBio) HiFi reads (~17-fold coverage) from the same gorilla individual (Kamilah) to the human genome reference (GRCh38) using minimap2 (Additional file 1: Table S5). We then assessed the read coverage over the region putatively deleted in gorilla using samtools [46] (v1.16.1, samtools view chr11:4,249,395–14,277,464) and bedtools [47] genomecov (v2.30.0). To further assess how these extracted reads aligned to the gorilla genome used herein as well as a newer gorilla genome assembly, we aligned the extracted reads (from samtools view) to each gorilla genome with minimap2, filtering for primary alignments with mapping quality ≥ 50 using samtools. The statistics of these alignments were assessed using bamtools [48] (v2.5.1).

Using the same parameters as the previous primate reference genome runs, we also computed synteny blocks between de novo genome assemblies of the same primate species (human, bonobo, chimpanzee, gorilla) using ntSynt and SibeliaZ. The human genome assembly was generated using GoldRush [49] with Oxford Nanopore long reads as input, while the other primate genome assemblies were generated using hifiasm [50] with PacBio HiFi long reads as input (Additional file 1: Table S6).

To assess ntSynt using genomes with higher sequence divergence, synteny blocks were computed between human (GRCh38), mouse (GRCm39) and rat (Rnor_6.0) reference genome builds with ntSynt (Additional file 1: Table S4). The approximate sequence divergence rates between these genome assemblies were assessed using Mash, as described for the primate tests. Consequently, ntSynt was run with *--divergence* 18.8 *--indel* 500,000, with all other parameters kept at the default values. Synteny blocks between these three genomes were also obtained from the SyntenyPortal web application (resolution = 150 kbp).

Finally, we detected synteny blocks between 11 bee genomes from the genus *Andrena* (Additional file 1: Table S7) using ntSynt. We ran ntSynt with *--indel* 100,000 *--divergence* 7.1, based on pairwise sequence divergence estimates from Mash.

All tests were run on a server with 144 Intel(R) Xeon(R) Gold 6254 CPU @ 3.1 GHz with 2.9 TB RAM using 12 threads. The python scripts used to assess the summary statistics of the output synteny blocks are available at https://github.com/bcgsc/ntSynt/tree/main/analysis_scripts, and example R scripts for generating the gggenomes [45] ribbon plots and chromosome sequence painting plots are available at https://github.com/bcgsc/ntSynt/tree/main/visualization_scripts.

Results

Pairwise synteny block detection using a simulated human genome

We first evaluated ntSynt and three comparators (SibeliaZ, halSynteny, SyRI) by generating pairwise synteny blocks between the human reference genome [51] and a simulated rearranged human genome, with uniformly distributed single-nucleotide variants (SNVs) and indels (small insertions/deletions) introduced at various rates [42, 43] (Fig. 2a–d, Additional file 1: Tables S8–10, Fig. S8–12). We simulated 20, 50–100 kbp indels (insertions and deletions), 2, 10–50 kbp translocations, and 20, 10–50 kbp inversions using SURVIVOR [42] (Additional file 1: Fig. S9). As this is a controlled experiment, the theoretical maximum synteny coverage possible for each tool is 100%. Compared to SibeliaZ, ntSynt generates substantially more contiguous synteny blocks, with block NG50 lengths (at least half of the genome covered by blocks ≥ this length) 1,525 to 9,514 times higher across the tested SNV and indel (variant) rates. In addition, the ntSynt synteny blocks have higher coverage, from 10.2 to 19.9% higher coverages for the lowest (0.1%) and highest (5.5%) variant rates, respectively (Fig. 2a–d). Compared to the pairwise synteny detection tools tested, ntSynt synteny blocks generally have lower block NG50 lengths, though these contiguities do not exceed the expected NG50 length of 61.7 Mbp given the ground truth (35.9–61.7 Mbp). SyRI and halSynteny behave differently, with 74.3–86.3 Mbp and 92.6–97.7 Mbp block NG50 lengths, respectively (Fig. 2a–d, Additional file 1: Fig. S10). ntSynt yields more correct synteny blocks than the pairwise comparator tools, as evidenced by accuracies over 97% across all divergences, compared to accuracies of 95–99% and 83–93% for halSynteny and SyRI, respectively (Additional file 1: Fig. S8, S10-11, Table S9). The lower accuracies for halSynteny and SyRI are also evident by higher than expected block NG50 lengths, as seen by the points falling above dotted line representing the ground truth in Fig. 2a and Fig. S10. In comparison, while the synteny blocks generated by SibeliaZ do have higher accuracies than ntSynt for the higher divergences (3.3–5.5%), the synteny blocks are highly fragmented, with block NG50 lengths than 20 kbp. While the other tools miss multiple structural rearrangement types, in these simulated tests, ntSynt only missed a small number of inversions (4 inversion errors across all 5 divergence runs) (Additional file 1: Fig. S12, Table S10). The synteny coverages for SyRI and halSynteny are over 98% for all tests, while the ntSynt coverages are over 99% for lower variant rates (0.1–1.1%), but 94–97% for the more divergent tests. Genomic regions that are not covered by ntSynt synteny blocks in these runs are mainly (> 95%) centromeric (Additional file 1: Table S11).

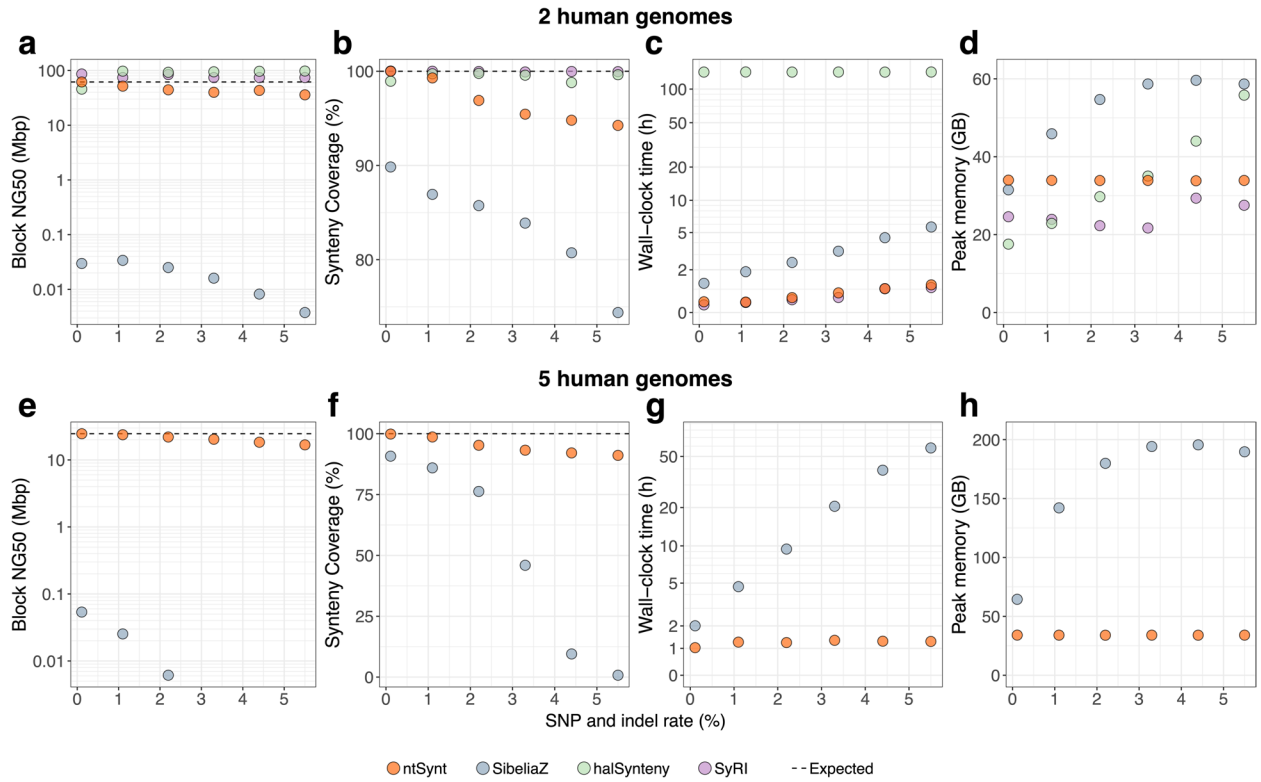

Fig. 2 Contiguity, coverage and benchmarking results from synteny block analysis between simulated human genome sequences. Rearrangements in the human genome were simulated using SURVIVOR [42], and SNVs and indels were introduced at increasing rates (ranging from 0.1 to 5.5%) using pIRS [43]. The top row plots (**a**–**d**) show the results of the pairwise comparisons between the human reference genome (T2T) and 1 rearranged human genome, while the bottom row (**e**–**h**) shows the results of a multi-genome synteny comparison between the human reference genome and 4 rearranged human genomes. halSynteny (green) and SyRI (purple) are only shown in (**a**–**d**), as they are pairwise synteny utilities, while the multi-genome synteny tools ntSynt (orange) and SibeliaZ (blue) are shown in all plots. The synteny block NG50 length and synteny coverage statistics (**a**, **b**, **e**, **f**) were averaged over all input genomes. The horizontal dashed lines represent the expected value for the corresponding statistic based on the ground truth. Plots **a**, **c**, **e**, **g** are shown in log-linear scale, while **b**, **d**, **f**, **h** are in linear scale. Benchmarks were averaged over triplicate runs, and the average wall-clock time and peak memory values plotted

In these pairwise tests, ntSynt ran 2.9 to 4.7-fold and 118.2 to 318.1-fold faster than SibeliaZ and halSynteny, respectively. SyRI was marginally (1.1–1.4 times) faster than ntSynt, with both tools running in less than 1.5 h across all variant rates tested. ntSynt and SyRI had fairly steady RAM usage (33.6–34.1 GB and 21.7–29.4 GB, respectively), but the RAM usage of SibeliaZ and halSynteny increased with higher variant rates, peaking at 59.6 GB and 60.9 GB, respectively.

Synteny blocks between multiple simulated human genome sequences

Next, we computed synteny blocks between multiple simulated input genomes with ntSynt and SibeliaZ (Fig. 2e–h, Additional file 1: Tables S12, 13). Five genomes were compared: the human reference genome, and four genomes with simulated nucleotide variations and rearrangements. In all runs, ntSynt generated synteny coverages between 91.1 and 99.9%, and block NG50 lengths between 16.8 and 24.6 Mbp, approaching the ground truth block NG50 length (24.6 Mbp). Similar to the pairwise comparisons aforementioned, ntSynt generates synteny blocks with block NG50 lengths 457 to 3,584-fold longer than SibeliaZ, and its synteny coverage was 9.1–90.8% higher. For variant rates above 3%, the SibeliaZ synteny blocks have less than 50% genome coverage, thus the associated block NG50 lengths could not be computed. Furthermore, in these tests ntSynt runs faster and has a lower memory footprint than SibeliaZ, with runtimes and peak memory usages ranging from 50 m to 2 h and 33.9 to 34.1 GB of RAM for ntSynt, in contrast to 1.7 h to 72.2 h and 64.4 to 195.6 GB of RAM for SibeliaZ.

Computing synteny blocks between four primate reference genomes

We computed synteny blocks between four primate reference genomes belonging to human, chimpanzee, bonobo,

and gorilla with ntSynt and SibeliaZ (Fig. 3a, Additional file 1: Tables S4, S14). Consistent with our simulated genome tests, ntSynt generated larger synteny blocks than SibeliaZ, with ntSynt blocks having an NG50 length of 7.6 Mbp, compared to 52.5 kbp for SibeliaZ. ntSynt also achieved higher synteny block coverage (92.2% vs. 88.6% for ntSynt and SibeliaZ, respectively), which can be readily visualized by zooming into chromosome 11, for example (Fig. 3b).

The outputs of ntSynt and SibeliaZ were also compared with synteny blocks from the SyntenyPortal web application (Fig. 3a). The synteny blocks computed using each approach are largely congruent, as evidenced by the consistent ribbon colors and transitions between the genome sequences for each chromosome (Fig. 3b). Although SyntenyPortal generates more contiguous synteny blocks than ntSynt (48.1 Mbp vs. 7.6 Mbp block NG50 lengths, respectively), this resource misses large variations between the genomes, as highlighted in Fig. 3b. Here, the ntSynt and SibeliaZ plots reveal a large (~10 Mbp) deletion around the 4 Mbp position of chromosome 11 in the gorilla genome. However, a single SyntenyPortal synteny block spans this large deletion. This deletion was recapitulated using pairwise minimap2 [29] alignments (Additional file 1: Fig. S13a). However, when gorilla long reads were aligned to the human reference genome (GRCh38), there were high-quality alignments found along the entire region identified as a putative deletion in gorilla by the synteny analysis, suggesting an error in the gorilla reference genome (Additional file 1: Table S5, Fig. S14). In this chromosome, there are additional examples of large indels missed by SyntenyPortal but detected using ntSynt (Additional file 1: Fig. S13a). The chromosome 11 SyntenyPortal blocks are only broken at a ~2 Mbp insertion in the gorilla genome, a rearrangement that ntSynt also detected (Additional file 1: Fig. S13b). Thus, unlike SyntenyPortal, ntSynt resolves large structural discrepancies in chromosome 11, providing insights into potential reference genome errors that some existing methods fail to reveal.

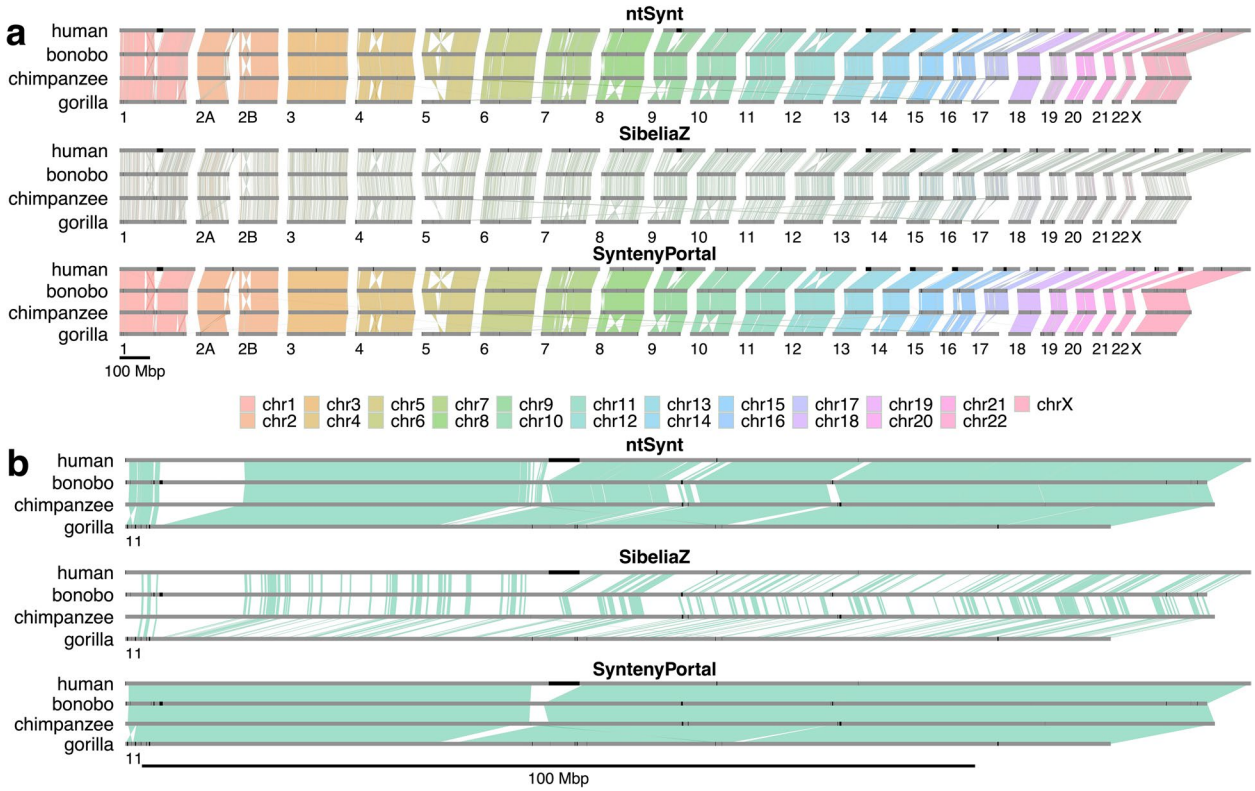

Fig. 3 Ribbon plots showing largely consistent synteny blocks computed between human, bonobo, chimpanzee, and gorilla genome sequences. Synteny blocks detected using ntSynt, SibeliaZ, and SyntenyPortal are shown. The chromosomes are indicated using grey bars, with gaps and centromeres in the reference chromosomes marked using black bars. Each ribbon is based on a synteny block ≥ 150 kbp that includes all 4 genomes. The ribbon colors represent the chromosome of the human genome in the synteny block, and each ribbon has a light grey border. The numbers below each plot indicate the chromosome in the other primate genomes, and the scales are indicated by the bar under each combined plot. Panel **a** shows the synteny blocks for the full genomes, and panel **b** zooms into chromosome 11. Twisted ribbons depict inverted sequence synteny while direct synteny blocks are represented by parallelograms

Computing synteny blocks between four primate genome assemblies

In a separate experiment, synteny blocks were computed between GoldRush [49] and hifiasm [50] human, bonobo, chimpanzee, and gorilla genome sequence assembly drafts using ntSynt and SibeliaZ (Additional file 1: Tables S6, 15), the only two utilities that can handle the simultaneous comparison of more than two genomes. The ntSynt four-genome synteny blocks had a coverage of 82.8% with 1.3 Mbp block NG50, while SibeliaZ generated synteny blocks with 79.1% coverage and 54.7 kbp block NG50 length.

Assessing synteny between human, mouse, and rat reference genomes

To assess the performance of ntSynt using empirical reference-grade genomes with higher sequence divergence, synteny blocks were computed between human, mouse, and rat reference genomes using ntSynt and SyntenyPortal (Fig. 4, Additional file 1: Tables S4 and S16). As SibeliaZ, which is only recommended for comparing closely-related genomes [31], yielded synteny blocks with less than 3% coverage for this experiment, it was omitted from the analysis. These genomes have high sequence divergence, with Mash [35] estimating pairwise sequence divergences up to 18.8%. The ntSynt synteny blocks between the three genomes had 79.2% coverage and 3.9 Mbp block NG50 length, while SyntenyPortal generated blocks with 89.8% coverage and 7.1 Mbp block NG50. As evidenced from the consistent colors and vertical nudges up and down in the chromosome sequence painting plots (Fig. 4), the synteny blocks computed by ntSynt and SyntenyPortal are largely consistent in terms of chromosome identities and block orientations. Notably, the majority of the large gaps seen in the ntSynt synteny blocks are seen in regions of centromere and gap sequences, as evidenced in human chromosomes 1, 4, 5, 11, and 12.

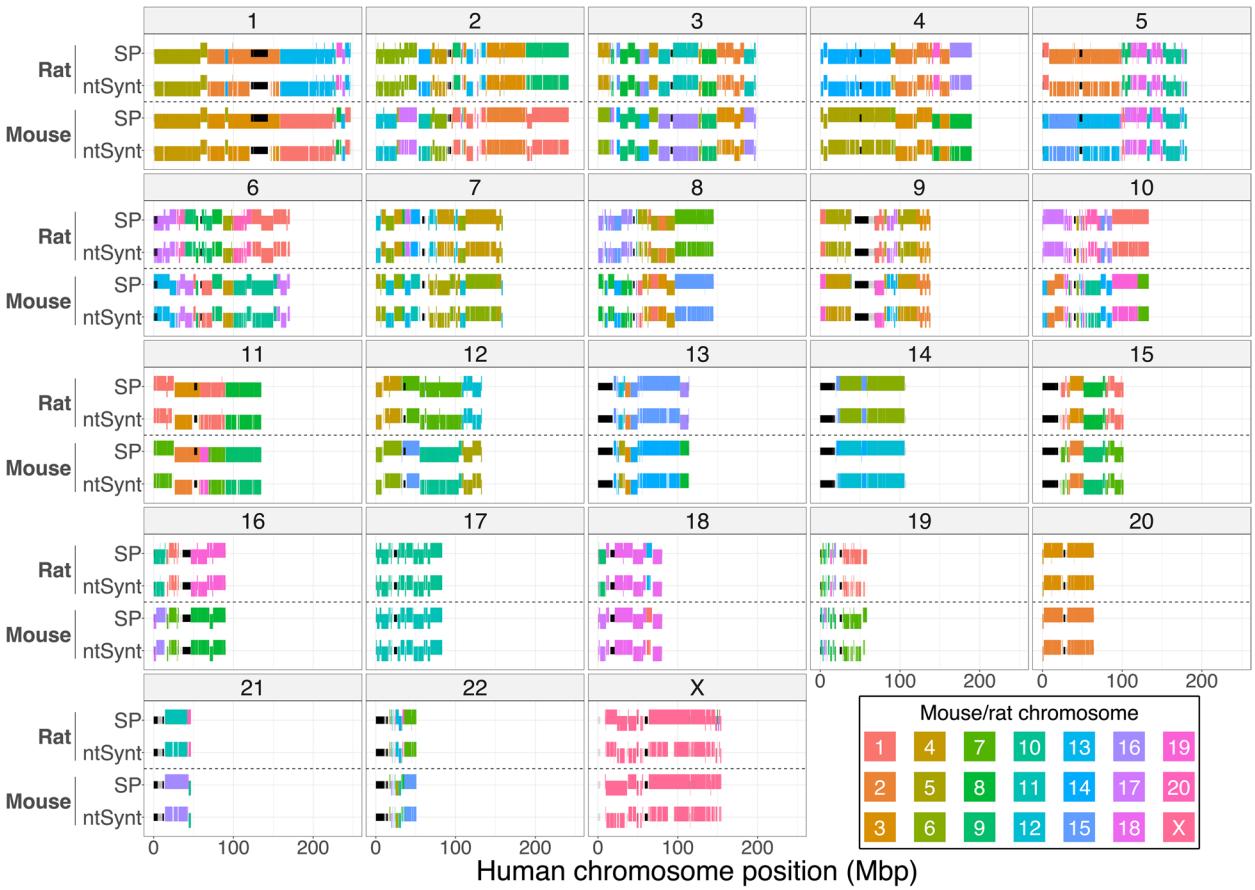

Fig. 4 Chromosome sequence painting plots comparing synteny blocks between human, mouse, and rat reference genomes. Synteny blocks were computed using ntSynt and SyntenyPortal (SP). Each facet is labeled with the corresponding human chromosome. The segments are colored by the chromosome in mouse or rat, respectively. Each segment represents a 3-genome synteny block in ntSynt or SyntenyPortal. The segments being nudged vertically up or down represent the forward or reverse orientation, respectively. The smaller black segments indicate gaps and centromere sequence in the human reference genome build

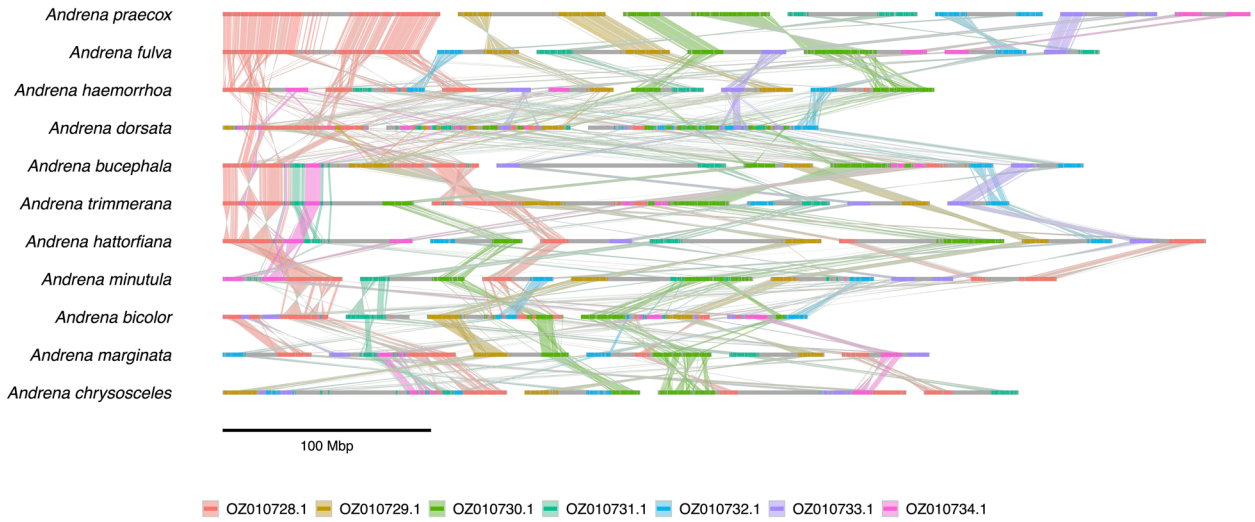

Fig. 5 Ribbon plots showing 11-way synteny blocks between *Andrena* bee genomes using ntSynt. The ribbons are colored based on the chromosome in the top genome, *Andrena praecox*. Twisted ribbons indicate inversions between the respective genomes, and each underlying chromosome is colored in grey

ntSynt generated synteny blocks between these three divergent genomes in 1.1 h using 34.2 GB of RAM (Additional file 1: Tables S16, 17).

Computing synteny between 11 bee genomes

Finally, synteny blocks were detected between 11 chromosome-level bee genomes from the genus *Andrena* using ntSynt (Fig. 5, Additional file 1: Table S18, Fig. S15) [52–62]. These assemblies, made available through the Darwin Tree of Life project [4], have a range of chromosome numbers (3–7), genome sizes (247–443 Mbp), and pairwise sequence divergences (1.9–7.1%). This 11-way comparison yielded synteny blocks with an N50 length of 481 kbp, covering 85% of the smallest genome with the ntSynt synteny blocks (Additional file 1: Table S18). The ribbon plot generated from the multi-genome synteny blocks computed by ntSynt reveal many rearrangements between the compared bee genomes. Interestingly, chromosome OZ010728.1 in *Andrena praecox* (colored in red) shows a high degree of synteny with *Andrena fulva*, but the chromosome is highly rearranged in the other genomes. These two species also form a clade in the phylogenetic tree generated using Mashtree [63] (Additional file 1: Fig. S15). In addition, as is evident by the purple coloring, chromosome OZ010733.1 in *A. praecox* has similar genomic content to a chromosome in *Andrena minutula* (placed second from the right), but the chromosome appears to have genomic expansions in *A. praecox*. ntSynt computed the synteny blocks in 14.7 m using at most 4 GB of RAM (Additional file 1: Table S18).

Discussion

ntSynt introduces a novel approach for detecting multi-genome synteny blocks by employing minimizer graphs to simultaneously map multiple genome sequences to one another. Unlike anchor-based synteny detection tools, ntSynt does not require annotations or precomputed markers, and utilizes full input genome sequences. With the introduction of various graph simplification and traversal algorithms, ntSynt builds upon the undirected minimizer graph approach introduced by ntJoin [36], adapting the principles for genome synteny detection. ntSynt also introduces a minimizer selection approach guided by a common Bloom filter [38], a probabilistic data structure used to store the *k*-mers found in all input genome assemblies. By only selecting minimizers that are shared across all input genomes, the resulting density of the minimizer sketch and graph is resilient to increased divergence between the genomes. These algorithms enable ntSynt's robustness when comparing multiple, divergent genomes. The effectiveness of this approach is demonstrated by the resulting megabase-range synteny block NG50 lengths with coverages over 92% when comparing four primate reference genomes, and the high synteny block coverage (85%) achieved when simultaneously comparing 11 bee genomes from the genus *Andrena* that exhibit extensive rearrangements. Moreover, ntSynt reliably recapitulates multiple well-characterized large-scale rearrangements between primate genomes, including the human chromosome 2 fusion [64] and a ~ 47 Mbp inversion in chromosome 12 [65], thereby demonstrating its accuracy and sensitivity in detecting evolutionary rearrangements.

Utilizing undirected minimizer graphs for multi-genome mapping, combined with the graph traversal and post-processing algorithms, allows for minor rearrangements and base-level differences between the input genome sequences to be tolerated within a synteny block, and substantial flexibility in the granularity of the ntSynt macrosynteny blocks. When using default parameters, ntSynt detected inversions, deletions, translocations, and inversions as small as 12 kbp in our controlled tests between genomes with simulated sequence divergence from 0.1 to 5.5%. The construction and traversal of these graphs are controlled by multiple parameters, including the merge, indel, and block size thresholds. These parameters have default values empirically determined based on a specified genome sequence divergence, but can be fine-tuned to refine the synteny blocks for particular research questions. For example, we observed that the largest synteny coverage gaps in the human-mouse-rat experiment were over centromere sequences and gaps; if a user desired collinear synteny blocks over these more distant regions to be merged, the merge threshold parameter could be increased. Generally, setting these thresholds higher allows for more contiguous synteny blocks and a broader view of the genome synteny. Conversely, lowering the thresholds will give a more granular picture of the genome synteny, making these parameters a balance between synteny block resolution and output block contiguity. These variable criteria for defining synteny blocks highlight why the block contiguity alone is not a perfect assessment statistic.

The other synteny block detection utilities tested have more limited parameters, constraining their flexibility in achieving these different resolutions. The minimum block size can be specified for SibeliaZ, halSynteny, and SyntenyPortal, but only the pairwise comparators halSynteny and SyRI have parameters for controlling the tolerated distance between anchoring alignments in a synteny block. None of the comparator tools have clear parameters controlling indel detection.

In addition to tunable parameters, ntSynt is also flexible to the number, divergence, and contiguity of the input genomes. ntSynt computes synteny blocks with megabase-scale contiguity and high coverage for closely-related genomes, such as those of primates, and more divergent genomes, like human and rodents, which diverged approximately 87 million years ago [66] and have a Mash-estimated divergence of ~18%. This robustness to variable divergences was also demonstrated by achieving synteny coverages above 91% when comparing five simulated human genomes, each with up to 5.5% simulated nucleotide sequence variant rates between them. This also demonstrates that using a mapping-based approach for synteny analysis, versus an annotation-guided anchor-based approach, can be effective for more divergent genomes, despite the contrary speculation reported in previous studies [9]. The robustness of the paradigm is further supported by the high multi-genome synteny block coverage (85% for the smallest genome) reached by ntSynt when detecting synteny between 11 bee genome assemblies from the *Andrena* genus, which vary in both chromosome number (3–7) and genome size (247–443 Mbp).

While halSynteny and SyRI produced synteny blocks with high coverage (> 98%) in the simulated genome tests, they are limited to pairwise genome comparisons. By design, SyRI utilizes the pairwise sequence aligner minimap2 [29] to generate alignments for computing synteny blocks. Progressive Cactus [30], the alignment engine behind halSynteny, did not scale well for the pairwise human comparisons, with each test requiring around 6 days to complete. While ntSynt was able to compare all 11 bee genomes in a single run, SyRI and halSynteny would need to be run 55 times to obtain pairwise comparisons between each of those genomes. This burden scales quadratically with the number of genomes, quickly becoming prohibitive for larger datasets, whereas ntSynt can always compare all input genomes simultaneously in a single, computationally efficient job. Additionally, SyRI requires compared genomes to have identical chromosome numbers, thus restricting syntenic analysis to less divergent genomes; for example, the karyotype of the bee genomes examined herein ranges from 3 to 7 chromosomes. Finally, both tools generate synteny blocks with higher than expected contiguities (as measured using the NG50 length metric), indicating that they fail to break the synteny blocks at all ground truth rearrangements.

Although SibeliaZ and SyntenyPortal compute multi-genome synteny blocks, they are less adaptable than ntSynt in the variety of compatible input genomes. SibeliaZ is designed for identifying locally collinear genome segments in closely related genomes, or those with an evolutionary distance of less than 0.09 substitutions per site to the most recent common ancestor [31]. This restriction leads to lower synteny coverage as the genomic sequence divergence increases, especially when comparing multiple genomes, as demonstrated in the simulated tests, and meant that SibeliaZ was not suitable for our human-mouse-rat experiment due to the high sequence divergence between those species' genomes (up to 18%). SibeliaZ's microsynteny focus was also evident from the contiguity of the synteny blocks, as ntSynt generated 1,400-times higher block NG50 lengths on average. On the other hand, SyntenyPortal can compare more divergent genomes, as seen by the > 89% coverage when comparing human, mouse and rat genomes, but can only use genome builds that are available on

the UCSC genome browser [32]. Thus, we could not use SyntenyPortal in our simulated tests, for comparing the GoldRush and hifiasm genome assemblies, or when analyzing the 11 bee genomes. While both SyntenyPortal and SibeliaZ are restricted in the synteny analyses that they can be used for, ntSynt can compare genome assemblies with a range of divergences.

While synteny tools aim to achieve as high synteny coverage as possible, the theoretical maximum value will not always be 100% depending on how the compared genomes have diverged. In our simulated human comparisons, we introduced large-scale rearrangements and uniformly distributed small nucleotide variants, so the maximum possible synteny coverage is 100%. However, for the comparisons of real genomes, this target number is not established, and can vary depending on how the genomes diverged as well as the definition of synteny blocks. Compared to SibeliaZ, ntSynt achieved higher synteny coverage for the primate experiments, and this coverage was more on par with that generated by SyntenyPortal, which is limited to using pre-computed sequence alignments for particular genome assembly builds. ntSynt's higher coverage in the simulated experiments compared to SibeliaZ combined with this consistency with SyntenyPortal supports that the higher synteny coverage obtained by ntSynt is largely correct, and demonstrates an important feature of the tool.

As well as its adaptability to synteny block granularity and input multiple genome sequences, ntSynt is also relatively fast and memory-efficient, largely due to its graph-based design and use of sequence minimizers. In all tests, comparing up to five ~3 Gbp genomes, ntSynt required at most 2 h and 34.2 GB of RAM. In comparison, SibeliaZ, the only command-line multi-genome synteny comparator, was more computationally expensive as the number and divergence of compared genomes increased, requiring at most 72.2 h and 195.6 GB RAM. Furthermore, the RAM usage of ntSynt remained between 33.6 and 34.2 GB in all large genome (~3 Gbp) tests and was 4.0 GB when comparing 11 bee genomes (247–443 Mbp), demonstrating that the required memory does not correlate with the number of input genomes. The most memory-intensive step was the creation of the common *k*-mer Bloom filter [38], the probabilistic data structure we used to store the *k*-mers occurring in all input genome assemblies (Additional file 1: Table S17, Supplementary Discussion). This memory efficiency is particularly important for enabling ntSynt's accessibility to a wide range of research environments.

In addition to its inflexibility toward the input genome assemblies, SyntenyPortal fails to break synteny blocks at various large indels. Examples can be seen in Fig. 3b, where multiple large indels were missed by SyntenyPortal, but detected by ntSynt, including a ~10 Mbp deletion in chromosome 11 of the gorilla genome. Further investigation into this large deletion using gorilla long sequencing reads suggested that the deletion is likely due to a misassembly in the older gorilla assembly release used herein. ntSynt breaking the synteny block at this region, and thus flagging it for further analysis, enabled the observation of this genome assembly issue. This feature may prove useful for evaluating the completeness and consistency of future genome assembly projects. Together, our results demonstrate that ntSynt can expose structural differences and/or potential reference errors that remain undetected by existing synteny tools.

Despite the contiguous and high coverage multi-genome synteny blocks achieved using ntSynt, the current implementation has a few limitations. Due to the nature of the minimizer mapping used, ntSynt will not detect duplications, which other tools, including SibeliaZ, can identify. Further, when comparing many highly divergent genomes, the mapping approach can lead to a lower synteny coverage. We do see a slight decrease in synteny coverage in our most divergent tests (e.g., human-mouse-rat) compared to the primate tests, although we also observe this decrease in the SyntenyPortal results. The impact of this issue could be mitigated by decreasing the *k*-mer size, which increases mapping sensitivity (Additional file 1: Fig. S16).

Conclusions

As we have demonstrated, ntSynt enables the alignment-free, scalable, and simultaneous computation of multi-genome synteny blocks for input genomes of various divergences. From deciphering pangenome structure within species to providing evolutionary insights between species, we expect the synteny blocks generated by ntSynt to lay the groundwork for comparative genomics analyses, and enable the genomics community to more effectively leverage the continuously expanding diversity of genome assembly resources across the tree of life.

Abbreviations

SNV Single-nucleotide variant
BF Bloom filter

Supplementary Information

The online version contains supplementary material available at https://doi.org/10.1186/s12915-025-02455-w.

Additional file 1: Fig. S1–S16. Table S1–S18.

Availability and requirements

Project name: ntSynt
Project home page: https://github.com/bcgsc/ntSynt
Operating system(s): Platform independent

Programming languages: Python, C++
Other requirements: Snakemake, btllib, samtools, seqtk
License: GPL-3.0
Any restrictions to use by non-academics: No

Authors' contributions

LC, IB and RLW conceptualized the study. LC, PK and JW implemented the algorithms. LC ran the tests, analysed the results, and wrote the manuscript. IB and RLW supervised the study. All authors contributed to editing the manuscript, and approved the final manuscript.

Funding

This study is supported by the Canadian Institutes of Health Research (CIHR) [PJT-183608, I.B., and PJT-203692, I.B.].

Data availability

The accessions for all reference genome assemblies analysed in this study can be found in Additional file 1: Tables S4, S6 and S7. The simulated rearranged human genomes are available on Zenodo (https://doi.org/10.5281/zenodo.10627623) [67]. ntSynt is freely available on GitHub (https://github.com/bcgsc/ntsynt) and the source code is archived on Zenodo (https://doi.org/10.5281/zenodo.16852738).

Declarations

Ethics approval and consent to participate

Not applicable.

Consent for publication

Not applicable.

Competing interests

The authors declare no competing interests.

Received: 17 October 2025 Accepted: 29 October 2025
Published online: 29 December 2025

References

1. Formenti G, Theissinger K, Fernandes C, Bista I, Bombarely A, Bleidorn C, et al. The era of reference genomes in conservation genomics. Trends in Ecology & Evolution Elsevier. 2022;37:197–202. https://doi.org/10.1016/j.tree.2021.11.008.
2. Wang T, Antonacci-Fulton L, Howe K, Lawson HA, Lucas JK, Phillippy AM, et al. The human pangenome project: a global resource to map genomic diversity. Nature. 2022;604:437–46. https://doi.org/10.1038/s41586-022-04601-8.
3. Lewin HA, Richards S, Lieberman Aiden E, Allende ML, Archibald JM, Bálint M, et al. The earth BioGenome project 2020: Starting the clock. Proceedings of the National Academy of Sciences. Natl Acad Sci. 2022;119:e2115635118.
4. Darwin Tree of Life Project Consortium. Sequence locally, think globally: the Darwin Tree of Life Project. Proceedings of the National Academy of Sciences. Natl Acad Sci. 2022;119:e2115642118.
5. Lian Q, Hüttel B, Walkemeier B, Mayjonade B, Lopez-Roques C, Gil L, et al. A pan-genome of 72 Arabidopsis thaliana accessions reveals a conserved genome structure throughout the global species range. PREPRINT (Version 1) available at Research Square. Research Square. 2023. https://doi.org/10.21203/rs.3.rs-2976609/v1.
6. Harling-Lee JD, Gorzynski J, Yebra G, Angus T, Fitzgerald JR, Freeman TC. A graph-based approach for the visualisation and analysis of bacterial pangenomes. BMC Bioinformatics. 2022;23:416. https://doi.org/10.1186/s12859-022-04898-2.
7. Beier S, Thomson NR. Panakeia - a universal tool for bacterial pangenome analysis. BMC Genomics. 2022;23:265. https://doi.org/10.1186/s12864-022-08303-3.
8. Goel M, Sun H, Jiao W-B, Schneeberger K. SyRI: finding genomic rearrangements and local sequence differences from whole-genome assemblies. Genome Biol. 2019;20:277. https://doi.org/10.1186/s13059-019-1911-0.
9. Lallemand T, Leduc M, Landès C, Rizzon C, Lerat E. An overview of duplicated gene detection methods: why the duplication mechanism has to be accounted for in their choice. Genes. 2020;11. https://doi.org/10.3390/genes11091046.
10. Drillon G, Carbone A, Fischer G, Public Library of Science. Synchro: a fast and easy tool to reconstruct and visualize synteny blocks along eukaryotic chromosomes. PLoS ONE. 2014;9:e92621. https://doi.org/10.1371/journal.pone.0092621.
11. Lee J, Hong W, Cho M, Sim M, Lee D, Ko Y, et al. Synteny Portal: a web-based application portal for synteny block analysis. Nucleic Acids Res. 2016;44:W35-40. https://doi.org/10.1093/nar/gkw310.
12. Haug-Baltzell A, Stephens SA, Davey S, Scheidegger CE, Lyons E. SynMap2 and SynMap3D: web-based whole-genome synteny browsers. Bioinformatics. 2017;33:2197–8. https://doi.org/10.1093/bioinformatics/btx144.
13. Robert NSM, Sarigol F, Zieger E, Simakov O. SYNPHONI: scale-free and phylogeny-aware reconstruction of synteny conservation and transformation across animal genomes. Bioinformatics. 2022;38:5434–6. https://doi.org/10.1093/bioinformatics/btac695.
14. Krasheninnikova K, Diekhans M, Armstrong J, Dievskii A, Paten B, O'Brien S. halSynteny: a fast, easy-to-use conserved synteny block construction method for multiple whole-genome alignments. GigaScience. Oxford University Press; 2020;9:giaa047.
15. Soderlund C, Bomhoff M, Nelson WM. SyMAP v3.4: a turnkey synteny system with application to plant genomes. Nucleic Acids Res. 2011;39:e68–e68. https://doi.org/10.1093/nar/gkr123.
16. Lv J, Havlak P, Putnam NH. Constraints on genes shape long-term conservation of macro-synteny in metazoan genomes. BMC Bioinformatics. 2011;12:S11. https://doi.org/10.1186/1471-2105-12-S9-S11.
17. Wright CJ, Stevens L, Mackintosh A, Lawniczak M, Blaxter M. Comparative genomics reveals the dynamics of chromosome evolution in Lepidoptera. Nature Ecology & Evolution. 2024;8:777–90. https://doi.org/10.1038/s41559-024-02329-4.
18. Simakov O, Bredeson J, Berkoff K, Marletaz F, Mitros T, Schultz DT, et al. Deeply conserved synteny and the evolution of metazoan chromosomes. Science Advances. American Association for the Advancement of Science. 2022;8:eabi5884. https://doi.org/10.1126/sciadv.abi5884.
19. Haas BJ, Delcher AL, Wortman JR, Salzberg SL. DAGchainer: a tool for mining segmental genome duplications and synteny. Bioinformatics. 2004;20:3643–6. https://doi.org/10.1093/bioinformatics/bth397.
20. Vergara IA, Chen N. Large synteny blocks revealed between *Caenorhabditis elegans* and *Caenorhabditis briggsae* genomes using OrthoCluster. BMC Genomics. 2010;11:516. https://doi.org/10.1186/1471-2164-11-516.
21. Nadeau JH, Taylor BA, National Acad Sciences. Lengths of chromosomal segments conserved since divergence of man and mouse. Proc Natl Acad Sci U S A. 1984;81:814–8.
22. Ried T, Schröck E, Ning Y, Wienberg J, Oxford University Press. Chromosome painting: a useful art. Hum Mol Genet. 1998;7:1619–26.
23. Wang Y, Tang H, DeBarry JD, Tan X, Li J, Wang X, et al. MCScanX: a toolkit for detection and evolutionary analysis of gene synteny and collinearity. Nucleic Acids Res. 2012;40:e49–e49. https://doi.org/10.1093/nar/gkr1293.
24. Pham SK, Pevzner PA. Drimm-synteny: decomposing genomes into evolutionary conserved segments. Bioinformatics. 2010;26:2509–16. https://doi.org/10.1093/bioinformatics/btq465.
25. Liu D, Hunt M, Tsai IJ. Inferring synteny between genome assemblies: a systematic evaluation. BMC Bioinformatics. 2018;19:26. https://doi.org/10.1186/s12859-018-2026-4.
26. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990;215:403–10. https://doi.org/10.1016/S0022-2836(05)80360-2.
27. Grabherr MG, Russell P, Meyer M, Mauceli E, Alföldi J, Di Palma F, et al. Genome-wide synteny through highly sensitive sequence alignment: satsuma. Bioinformatics. 2010;26:1145–51. https://doi.org/10.1093/bioinformatics/btq102.
28. Marçais G, Delcher AL, Phillippy AM, Coston R, Salzberg SL, Zimin A, et al. MUMmer4: a fast and versatile genome alignment system. PLoS Comput Biol. 2018;14:e1005944. https://doi.org/10.1371/journal.pcbi.1005944.

29. Li H. Minimap2: pairwise alignment for nucleotide sequences. Bioinformatics. 2018;34:3094–100. https://doi.org/10.1093/bioinformatics/bty191.
30. Armstrong J, Hickey G, Diekhans M, Fiddes IT, Novak AM, Deran A, et al. Progressive cactus is a multiple-genome aligner for the thousand-genome era. Nature. 2020;587:246–51. https://doi.org/10.1038/s41586-020-2871-y.
31. Minkin I, Medvedev P. Scalable multiple whole-genome alignment and locally collinear block construction with SibeliaZ. Nat Commun. 2020;11:6327. https://doi.org/10.1038/s41467-020-19777-8.
32. Kent WJ, Sugnet CW, Furey TS, Roskin KM, Pringle TH, Zahler AM, et al. The human genome browser at UCSC. Genome Research Cold Spring Harbor Lab. 2002;12:996–1006.
33. Ma J, Zhang L, Suh BB, Raney BJ, Burhans RC, Kent WJ, et al. Reconstructing contiguous regions of an ancestral genome. Genome Research. Cold Spring Harbor Lab; 2006;16:1557–65.
34. Roberts M, Hayes W, Hunt BR, Mount SM, Yorke JA. Reducing storage requirements for biological sequence comparison. Bioinformatics. 2004;20:3363–9. https://doi.org/10.1093/bioinformatics/bth408. England.
35. Ondov BD, Treangen TJ, Melsted P, Mallonee AB, Bergman NH, Koren S, et al. Mash: fast genome and metagenome distance estimation using MinHash. Genome biology BioMed Central. 2016;17:1–14.
36. Coombe L, Nikolić V, Chu J, Birol I, Warren RL. ntJoin: fast and lightweight assembly-guided scaffolding using minimizer graphs. Bioinformatics. 2020;36:3885–7. https://doi.org/10.1093/bioinformatics/btaa253.
37. Coombe L, Warren RL, Wong J, Nikolic V, Birol I, John Wiley & Sons, Ltd. Ntlink: a toolkit for de novo genome assembly scaffolding and mapping using long reads. Curr Protoc. 2023;3:e733. https://doi.org/10.1002/cpz1.733.
38. Bloom BH. Space/time trade-offs in hash coding with allowable errors. Communications of the ACM. ACM New York, NY, USA; 1970;13:422–6.
39. Nikolić V, Kazemi P, Coombe L, Wong J, Afshinfard A, Chu J, et al. Btllib: a C++ library with Python interface for efficient genomic sequence processing. J Open Source Softw. 2022;7:4720.
40. Kazemi P, Wong J, Nikolić V, Mohamadi H, Warren RL, Birol I. NtHash2: recursive spaced seed hashing for nucleotide sequences. Bioinformatics. 2022;38:4812–3. https://doi.org/10.1093/bioinformatics/btac564.
41. Mölder F, Jablonski K, Letcher B, Hall M, Tomkins-Tinch C, Sochat V, et al. Sustainable data analysis with Snakemake [version 2; peer review: 2 approved]. F1000Research. 2021;10. https://doi.org/10.12688/f1000research.29032.2.
42. Jeffares DC, Jolly C, Hoti M, Speed D, Shaw L, Rallis C, et al. Transient structural variations have strong effects on quantitative traits and reproductive isolation in fission yeast. Nat Commun. 2017;8:14061. https://doi.org/10.1038/ncomms14061.
43. Hu X, Yuan J, Shi Y, Lu J, Liu B, Li Z, et al. pIRS: profile-based Illumina pair-end reads simulator. Bioinformatics. 2012;28:1533–5. https://doi.org/10.1093/bioinformatics/bts187.
44. Kolmogorov M, Armstrong J, Raney BJ, Streeter I, Dunn M, Yang F, et al. Chromosome assembly of large and complex genomes using multiple references. Genome research Cold Spring Harbor Lab. 2018;28:1720–32.
45. Hackl T, Ankenbrand MJ, Adrichem B van. gggenomes: a grammar of graphics for comparative genomics. 2023. https://github.com/thackl/gggenomes.
46. Danecek P, Bonfield JK, Liddle J, Marshall J, Ohan V, Pollard MO, et al. Twelve years of SAMtools and BCFtools. Gigascience. United States. 2021;10. https://doi.org/10.1093/gigascience/giab008.
47. Quinlan AR, Hall IM. BEDtools: a flexible suite of utilities for comparing genomic features. Bioinformatics. 2010;26:841–2. https://doi.org/10.1093/bioinformatics/btq033.
48. Barnett DW, Garrison EK, Quinlan AR, Strömberg MP, Marth GT. BamTools: a C++ API and toolkit for analyzing and managing BAM files. Bioinformatics. 2011;27:1691–2. https://doi.org/10.1093/bioinformatics/btr174.
49. Wong J, Coombe L, Nikolić V, Zhang E, Nip KM, Sidhu P, et al. Linear time complexity de novo long read genome assembly with GoldRush. Nat Commun. 2023;14:2906. https://doi.org/10.1038/s41467-023-38716-x.
50. Cheng H, Concepcion GT, Feng X, Zhang H, Li H. Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm. Nat Methods. 2021;18:170–5. https://doi.org/10.1038/s41592-020-01056-5.
51. Nurk S, Koren S, Rhie A, Rautiainen M, Bzikadze AV, Mikheenko A, et al. The complete sequence of a human genome. Science. American Association for the Advancement of Science. 2022;376:44–53. https://doi.org/10.1126/science.abj6987.
52. Falk S, Mulley JF, of Oxford U, Lab WWGA, of Life WSIT, Darwin Tree of Life Consortium. The genome sequence of the short-fringed mining bee, Andrena dorsata (Kirby, 1802). Wellcome Open Research. 2023;8:373.
53. Crowley LM, University of Oxford and Wytham Woods Genome Acquisition Lab, Darwin Tree of Life Barcoding collective, Wellcome Sanger Institute Tree of Life Management S and L team, Wellcome Sanger Institute Scientific Operations: Sequencing Operations, Wellcome Sanger Institute Tree of Life Core Informatics team, et al. The genome sequence of the Hawthorn Mining Bee, Andrena chrysosceles (Kirby, 1802). Wellcome Open Research. F1000 Research Limited London, UK; 2025;10:100.
54. Crowley LM, of Oxford U, Lab WWGA, of Life WSIT, Darwin Tree of Life Consortium. The genome sequence of the big-headed mining bee, Andrena bucephala (Stephens, 1846). Wellcome Open Research. 2024;9:111.
55. Baker E, Crowley LM, Falk S, University of Oxford and Wytham Woods Genome Acquisition Lab, Darwin Tree of Life Barcoding collective, Wellcome Sanger Institute Tree of Life Management S and L team, et al. The genome sequence of Trimmer's Mining Bee, Andrena trimmerana (Kirby, 1802). Wellcome Open Research. F1000 Research Limited London, UK; 2024;9:691.
56. Falk S, Monks J, of Oxford U, Lab WWGA, of Life WSIT, Darwin Tree of Life Consortium. The genome sequence of Gwynne's mining bee, Andrena bicolor Fabricius, 1775. Wellcome Open Research. 2024;9:140.
57. Mitchell R, Monks J, of Life WSIT, Darwin Tree of Life Consortium. The genome sequence of the Small Scabious Mining Bee, Andrena marginata Fabricius, 1776. Wellcome Open Research. 2024;9:447.
58. Crowley LM, Mulley JF, of Oxford U, Lab WWGA, of Life WSIT, Darwin Tree of Life Consortium. The genome sequence of the Tawny Mining Bee, Andrena fulva (Müller, 1766). Wellcome Open Research. 2023;8:258.
59. Crowley LM, University of Oxford and Wytham Woods Genome Acquisition Lab, Darwin Tree of Life Barcoding collective, Wellcome Sanger Institute Tree of Life Management S and L team, Wellcome Sanger Institute Scientific Operations: Sequencing Operations, Wellcome Sanger Institute Tree of Life Core Informatics team, et al. The genome sequence of the Small Sallow Mining Bee, Andrena praecox (Scopoli, 1763). Wellcome Open Research. F1000 Research Limited London, UK; 2025;10:290.
60. Falk S, Tan K-T, of Oxford U, Lab WWGA, of Life WSIT, Darwin Tree of Life Consortium. The genome sequence of the Large Scabious Mining Bee, Andrena hattorfiana (Fabricius, 1775). Wellcome Open Research. 2023;8:224.
61. Crowley LM, of Oxford U, Lab WWGA, of Life WSIT, Darwin Tree of Life Consortium. The genome sequence of the Orange-tailed Mining Bee, *Andrena haemorrhoa* (Fabricius, 1781). Wellcome Open Res. 2023;8:396.
62. Falk S, University of Oxford and Wytham Woods Genome Acquisition Lab, Darwin Tree of Life Barcoding collective, Wellcome Sanger Institute Tree of Life programme, Wellcome Sanger Institute Scientific Operations: DNA Pipelines collective, Tree of Life Core Informatics collective, et al. The genome sequence of the common mini-mining bee Andrena minutula (Kirby, 1802). Wellcome Open Research. F1000 Research Limited London, UK; 2022;7:300.
63. Katz LS, Griswold T, Morrison SS, Caravas JA, Zhang S, den Bakker HC, et al. Mashtree: a rapid comparison of whole genome sequence files. J Open Source Softw. 2019;4(44):1762.
64. Shao Y, Zhou L, Li F, Zhao L, Zhang B-L, Shao F, et al. Phylogenomic analyses provide insights into primate evolution. Science. American Association for the Advancement of Science. 2023;380:913–24. https://doi.org/10.1126/science.abn6919.
65. Mao Y, Catacchio CR, Hillier LW, Porubsky D, Li R, Sulovari A, et al. A high-quality bonobo genome refines the analysis of hominid evolution. Nature. 2021;594:77–81. https://doi.org/10.1038/s41586-021-03519-x.
66. Kumar S, Suleski M, Craig JM, Kasprowicz AE, Sanderford M, Li M, et al. TimeTree 5: an expanded resource for species divergence times. Mol Biol Evol. 2022;39:msac174. https://doi.org/10.1093/molbev/msac174.
67. Coombe L, Kazemi P, Wong J, Birol I, Warren RL. Simulated assemblies for "Multi-genome synteny detection using minimizer graph mappings". Zenodo; 2024. https://zenodo.org/doi/10.5281/zenodo.10627622.

Publisher's Note