# Dot-to-dot user manual

Version 1.0.p4

Dot-to-dot accepts parameters via command line and configuration file. When a parameter is specified with both methods the command line has the priority.
In order to prevent errors due to misspells, the parameters of the configuration file as well as the value of the variable FilterType and AllowOverlap are case insensitive.

## Options

Command line option list (in {} the corresponding option in the configuration file:

- **-s, --sequence {Sequence}**:
  sequence file name (accept fasta/multifasta/fastq). The file format is detected inspecting the sequences' header. The file extension is ignored.
- **-c, --config**: [Command line only]
  configuration file name (required)
- **-o, --output {Outfile}**:
  output file name. The output format is determined according to the file extension (dot/bed). In absence of a supported extension the suffix .dot is added and the output format is selected accordingly. If this option is not specified the standard output is used. If the output file cannot be opened the standard output is used in its place.
- **-l, --minmotif {MinMotifLen}**: [default=2]
  minimum size in bp of the motif sequence
- **-L, --maxmotif {MaxMotifLen}**: [default=30]
  maximum size in bp of the motif sequence
- **-m, --minmatch {MinMatch}**: [range (0,1) default=1]
  Minimum overall matching score normalized in the range [0,1]
- **-G, --maxgaps {MaxGaps}**: [default=0]
  maximum number of mismatches in a motif (expressed in bp). This value is automatically turned into a minimum threshold by subtracting it to the actual motif length. As a result using matching weights lower than 1 can make this filter more severe than expected.
- **-I, --maxinsert {MaxInsert}**: [default=0]
  maximum insert size expressed in bp. During searching dot-to-dot automatically limits this value to be lower than the motif length.
- **-t, --threads {Threads}**: [default=1]
  number of threads. Every sequence is analyzed using just one thread, thus the number of threads should be lower or equal to the number of sequences. Since each thread allocates its own dot matrix the amount of used memory depends on the sequences length. For small sequences (like reads) a large number of threads (as large as the number of cores of the machine) can speedup the computation using a limited amount of memory. For analyzing an entire genome it could be better to use a moderate parallelism. A conservative setting (tailored on the

chromosome 1 of hg38) could roughly be using one thread every 8Gb of RAM.

- **-v, --verbose**:
  print on the standard error the id of a sequence once loaded
- **-V, --version**:
  print dot-to-dot version and exit
- **-h, --help**:
  print a help page and exit

Notice: the thresholds defined with –m and –G are merged into a single threshold according to the following formula.

$$score >= Threshold = \max(|\text{Motif}| - \text{MaxGaps}, \lfloor |\text{Motif}| * \text{MinMatch} \rfloor)$$

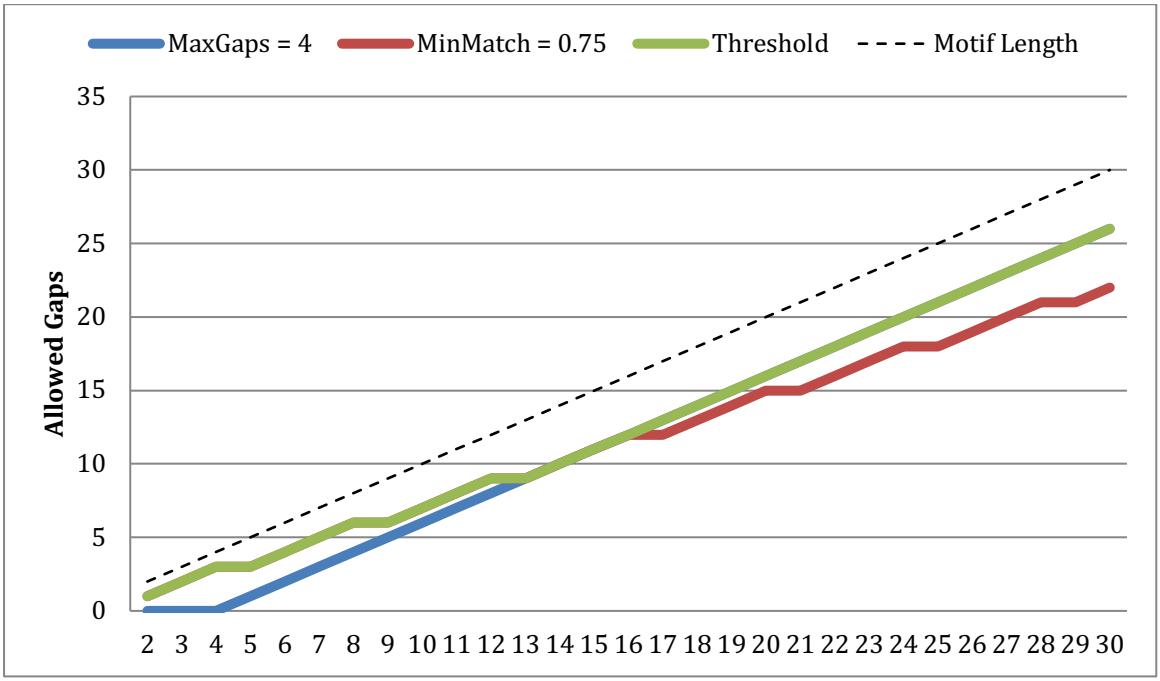


Figure 1: Threshold score varying the motif length

## Filtering (only via configuration file)

Filtering is controlled by means of the following variables specified in the configuration file:

- **FilterType**: [default=NONE] sets the level of filtering from disabled to the most aggressive
  - **NONE**: disable filtering
  - **THRESHOLD**: filters-out only: too small TRs (controlled via MinTRLen) and repeats with an unsatisfactory purity (controlled via MinPurity). All the other filtering strategies apply threshold filtering first.
  - **LIGHT**: retains tandem repeats of an overlapping group that match at least one of the three statistics (longest, most pure, shortest motif length)

    - **FAIR**: counts for each tandem repeat of an overlapping group the number of matched statistics (i.e. longest and most pure counts as two) and retains repeats with the highest count in the group.
    - **HEAVY**: it is the most stringent filtering. Each tandem repeat of an overlapping group is scored with the sum of its purity plus its length ratio (namely its length divided by the maximum length within the group). Only repeats with score over threshold are retained. Threshold is computed subtracting a tolerance value (controlled via the Tolerance parameter) from the maximal score.
- **MinTRLen**: [default=12] minimum length of a tandem repeat to be included in the output.
- **MinPurity**: [range (0,1) default=0.1] minimum purity of a tandem repeat to be included in the output.
- **Tolerance:** [range (0,1) default=0] (used only with heavy filtering). Set a degree of tolerance to be accepted using heavy filtering. This parameter ranges between 0 and 1 where the lower value the most stringent the filtering is.
- **AllowOverlap**: [range (Y/N) default=Y]. Controls the possibility of overlapping results in the final output. This filter is the last applied in the filtering process. For each pair of consecutive overlapping tandem repeats the purest is retained. In case of tie the longest is preferred. If length still does not break the tie, in absence of further discriminants, we discard the second repeat. Although arbitrary, this policy contributes to reduce the probability of overlapping with subsequent tandem repeats.

## Matching weights (only via configuration file)

Matching weights are controlled only via configuration file. Dot-to-dot allows scoring also mismatching characters. Scores are floating point values ranging between 0 and 1.

To add a score insert a two character long string specifying the two matching characters followed by = and the score. Undefined pairs are considered as having score equal to 0. The weight matrix is symmetric. This means that defining AC = x implies CA = x.

We discourage the use of a score for the NN pair since the genome contains very long stretches of Ns that would be considered as tandem repeats.

The input sequences and the weights are not capitalized. If you have sequences containing mixed lowercase and uppercase characters it is could be desired to add weights for the combinations (Aa, aA, cC, Cc, Gg, gG, Tt, tT)

Match scores are restricted only to the DNA alphabet (both lowercase and uppercase) including the special characters N and n. Combinations including other characters are ignored.

## Output file format (.dot)

The dot format is a tab-delimited file containing several columns. The first line (starting with the symbol #) is a header describing the columns content.

- **Seqid**: short string describing the sequence. It is extracted from the fasta/fastq header. For brevity it is truncated to the first blank character in the sequence header
- **Start coordinate**: initial position of the TR in the sequence (coordinates are 1-based in this format)
- **End coordinate**: ending position of the TR in the sequence (coordinates are 1-based in this format)
- **Copy number**: (integer) number of copies of the repeated motif (not including insertions and the last motif for fractional TRs)
- **Motif length**: length in bp of the repeated motif
- **Purity**: purity value computed as the ratio between the number of matches and the length of the TR
    - gaps are considered as mismatches regardless their content
    - for fractional TRs the last instance is considered as matching
- **Sequence**: space segmented TR sequence. Except for the last segment, a length lower than the motif length must be interpreted as an insertion.

In order to avoid ambiguity, the tandem repeats are reported in the same order of the input sequences and within a sequence they are sorted from the smallest to the highest coordinate.

### Output file format (.bed)

The bed file follows the UCSC specifications (https://genome.ucsc.edu/FAQ/FAQformat.html#format1). Only the first three mandatory fields are reported (chromosome, start position and end position). Coordinates are 0-based. Following the UCSC specification the bed file contains a header starting with the character #.
As for the dot format the chromosome is extracted from the fasta/fastq header. For brevity it is truncated to the first blank character in the sequence header.
In order to avoid ambiguity, the tandem repeats are reported in the same order of the input sequences and within a sequence they are sorted from the smallest to the highest coordinate.

### Tuning multithreading

Dot2dot implements multithreading providing each thread with a sequence. This is necessary because some data structures for filtering are updated while searching for TRs, thus breaking the sequence would have introduced local changes able to reflect on the output.
Once a thread has finished its work on a sequence, it attempts to store results on the output file. In order to keep the same sequence ordering as the input file, threads wait until results on the previous sequences have been written. This behavior makes Dot2dot running time to depend on the distribution of the lengths of the input sequences (the worst case is when the same thread tends to be assigned sequences much longer than the others). On the other hand, however, this limits the use of RAM avoiding to keep too many results pending to be written.

Threads implement a mechanism to reuse RAM. After computation a thread does not free the memory used for the sequence and for storing results, but reuses it reallocating only if more space is needed. Although this might seem that the Dot2dot memory usage becomes larger than necessary, the peak of allocated RAM does not change, instead, fragmentation is greatly reduced and the number of calls to the malloc () function are minimized. If necessary, the memory management of modern OSs makes the unnecessary memory to be moved in the swap without affecting the system performances.
In order to decide the number of threads to use, it is recommended to think in terms of the longer expected input sequence. Given a sequence of length L, a thread needs to maintain a vector of L floats (32bits) for each character in the sequence alphabet that in the worst case of lowercase and uppercase characters has cardinality 10 (A,C,G,T,N,a,c,g,t,n). A further vector of chars is needed to store the sequence itself, while a vector of L pointers is necessary for providing direct positional access to the matrix. Results are stored in a packed data structure that, in general, does not require more than a further 15% of space.
As a practical reference, hg38 is probably the genome with the longest sequences one can deal with. Given a version in which there were only uppercase characters, Dot2dot run with 8 threads using a peak of about 43Gb of RAM. Bigger genomes with shorter sequences used much less memory.
Dealing with reads, memory is never an issue, but the number of threads needs to be tuned to avoid (or minimize) waiting time of threads. If the sequences are all the same size, the threads tend to take the same time and thus they tend to behave as in a sort of round robin. In this case, a number of threads of the same order of the number of processor cores is appropriate.
When sequence lengths are not smoothly distributed, increasing the number of threads (to reach two times the number of physical cores) helps to mitigate blowup caused by waiting long sequences to complete.

## The Issues

Report bugs and issues to filippo.geraci@iit.cnr.it