Atom-Parser Documentation

Benjamin Schrauf

October 14, 2017

Overview

This python script prepares the input data for the DFTB+ electron transport density functional theory code. The program takes a molecule, given in the form of atom positions, as input. Also provided are the maximum distances at which the atoms interact and the contact atoms through which electric current is introduced to the molecule. Using this information, the program divides the atoms into groups ("bins"). Each bin's atoms may only be in contact with atoms from at most two other bins. To satisfy this contraint, the program must generate an equivalent one-dimensional structure by merging together the appropriate bins and contacts until only two contacts are left.

1 Task description

The DFTB+ code uses tridiagonal Hamiltonian matrices to solve molecular electron transport problems. To generate a tridiagnal Hamiltonian matrix, the atoms need to be sorted into groups ("bins") in such a way that the atoms from one bin only interact with atoms from at most two other bins. Such an arrangement is only possible if the bins are ordered in a one-dimensional chain, with two electrical contacts on each end. Thus, the sorting algorithm must be able to partition the atoms of an arbitrary molecule with many contacts into an equivalent one-dimensional array of bins with two electrical contacts forming the ends.

Because the DFTB+ code slows down dramatically for larger bin sizes, the sorting algorithm should also attempt to create bins of similar size.

The Atom-Parser algorithm is supposed to be a reliable data preparation tool for the users of the DFTB+ code. Therefore, the output of the partitioning process must be automatically checked for validity. This includes checking whether the atoms in each bin only interact with atoms in the two neighboring bins, checking whether all atoms of the molecule have been sorted, and checking for duplicate atoms within the sorted atoms.

2 Algorithm

The program assigns an index to each atom of the input molecule, and associates this index with a position by a list. The program creates a list of atom indices for the atoms in each electrical contact, as well as a list of atom indices for the device atoms. Finally, the program generates a dictionary that defines the maximum interaction distances for any given pair of elements present in the molecule. Beyond this threshhold, there is assumed to be no interaction between two atoms.

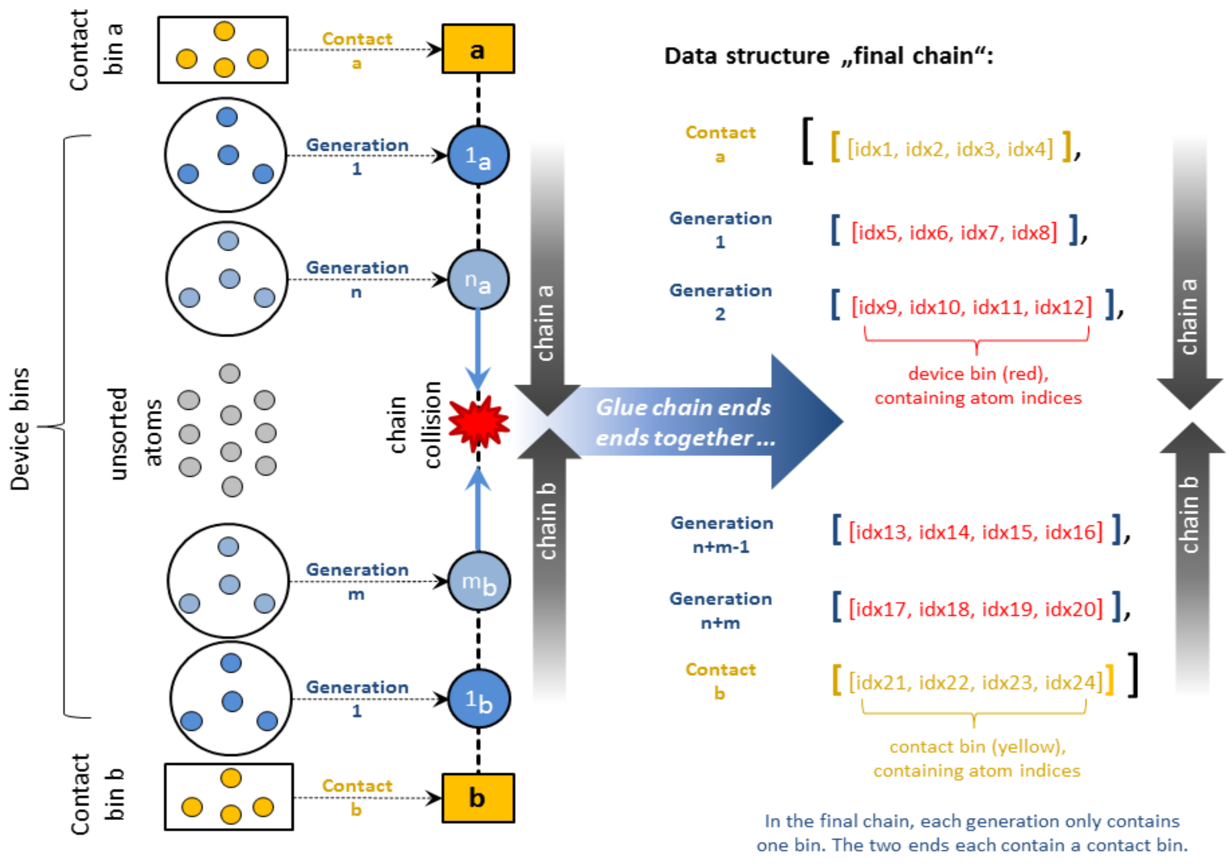
2.1 Data preparation

First the program generates two $n \times n$ -matrices, where n is the number of atoms in the input molecule. In these matrices, each column and row is associated with an atom index. The first matrix (dist_mtrx) gives the distances between each pair of atoms in the molecule, the second matrix (interact_mtrx) is composed of truth values that indicate whether any given pair of atoms is within interaction distance.

2.2 Partitioning atoms into bins

We will examine the sorting process for a simple test case: a one-dimensional molecule (see figure 1). The atoms are sorted into bins starting from all contacts simultaneously. In this case, there are only two contacts, a and b, at the two ends of the molecule. Starting from these two contacts (depicted as black boxes with orange atoms), the program generates a one-dimensional chain of bins (black circles with blue atoms), with the atoms in each bin only interacting with atoms in the previous and the next bin of the chain. In this manner, two chains a and b are generated, starting from the two contacts. Each bin in the chain belongs to a "generation", which is given by the number of steps a bin is away from the contact bin, at which the chain originates.

At some point, the tips of the chains collide. In the simple one dimensional case, a collision of the chains implies that all atoms have been sorted. In a last step, the ends of the two chains are now "glued" together to form the data structure "final_chain".



Contact bin a
Contact a
a
Device bins
Generation 1
1a
Generation n
na
unsorted atoms
chain collision
chain a
chain b
Glue chain ends ends together...
Generation m
mb
Generation 1
1b
Contact bin b
Contact b
b
Data structure „final chain“:
Contact a
[[idx1, idx2, idx3, idx4]],
Generation 1
[[idx5, idx6, idx7, idx8]],
Generation 2
[[idx9, idx10, idx11, idx12]],
device bin (red), containing atom indices
Generation n+m-1
[[idx13, idx14, idx15, idx16]],
Generation n+m
[[idx17, idx18, idx19, idx20]],
Contact b
[[idx21, idx22, idx23, idx24]]]
contact bin (yellow), containing atom indices
In the final chain, each generation only contains one bin. The two ends each contain a contact bin.

Figure 1: Generating the "final chain" data structure. Left side: Molecule with colored dots representing atoms. The black circles are the bins the atoms are sorted into, where atoms in one bin only interact with the atoms in the two adjacent bins. Center: Sorted chains a and b, consisting of bins. Right side: Data structure "final_chain", consisting of a nested list containing atom indices. The "final_chain" list is created by glueing together the ends of the two chains a and b.

2.3 Merging chains

In general, the input molecules will not have a one-dimensional structure. In order to create the desired one-dimensional ”final_chain” data structure, we must merge pairs of chains together until only two colliding chains are left, which can then be glued together into a ”final_chain” as described above. The merging process is shown in figures 2 and 3, using the simple example of a T-junction molecule with three contacts a, b and c.

In this algorithm, the merging happens concurrent to the partitioning process. Pairs of chains are merged as soon as they collide. This approach has the disadvantage of not being able to minimize the size of the resulting bins, since chains will be merged together irrespective of their size. A more optimized approach would first finish the partitioning before doing any merging. However, this approach would have been beyond the scope of the project, and for this reason was avoided.

Before merging, duplicate atoms must be removed from the chain tips. In particularly difficult cases, there can be multiple collisions between chains in a single generation. For this reason, the program finds all collisions in a given generation before attempting to merge chains.

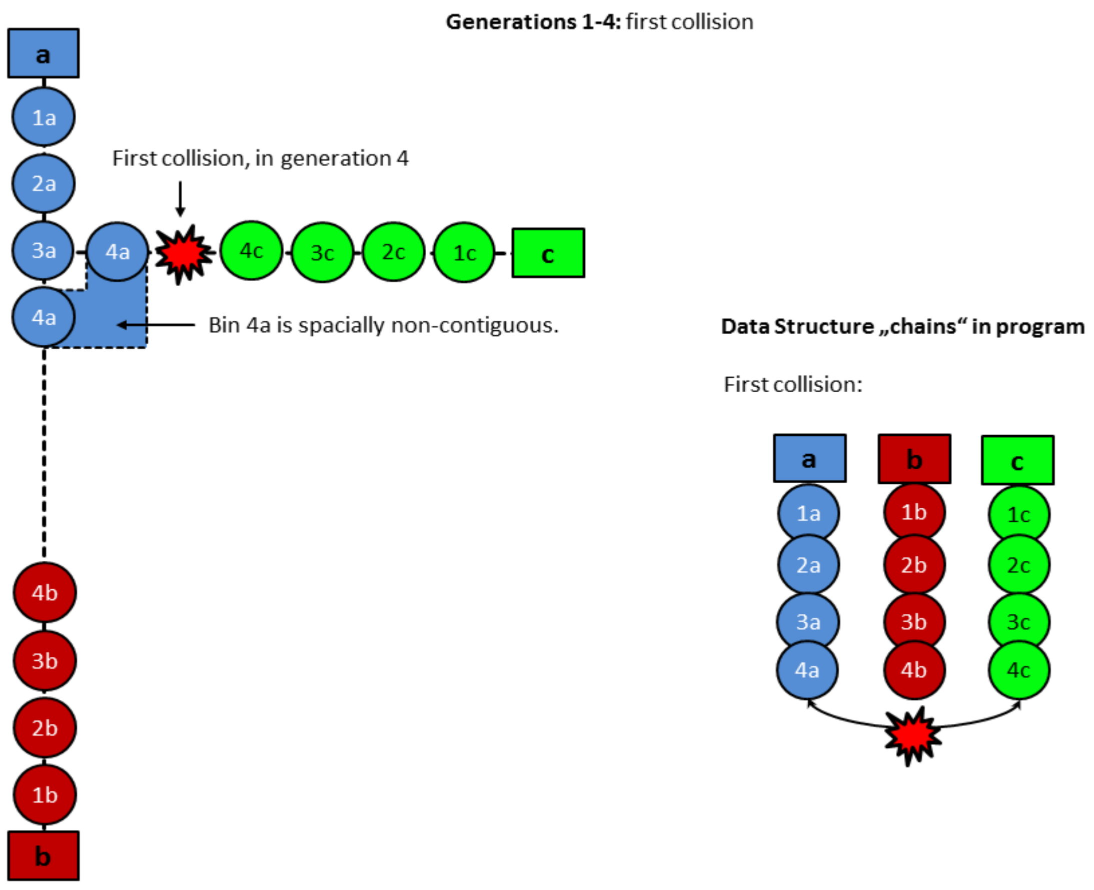

Generations 1-4: first collision
a
1a
2a
3a
4a
4a
First collision, in generation 4
4c
3c
2c
1c
c
Bin 4a is spacially non-contiguous.
4b
3b
2b
1b
b
Data Structure „chains“ in program
First collision:
a
b
c
1a
2a
3a
4a
1b
2b
3b
4b
1c
2c
3c
4c

Figure 2: Partitioning of a T-junction molecule. Left side: Molecule during partitioning process, after collision of chains a and c. Note that bin 4a is spacially non-contiguous: It consists of two groups of atoms that are outside of mutual interaction range. Such bins are generated because the partitioning process spreads like a wavefront throughout the molecule. Right side: Data structure ”chains” in memory.

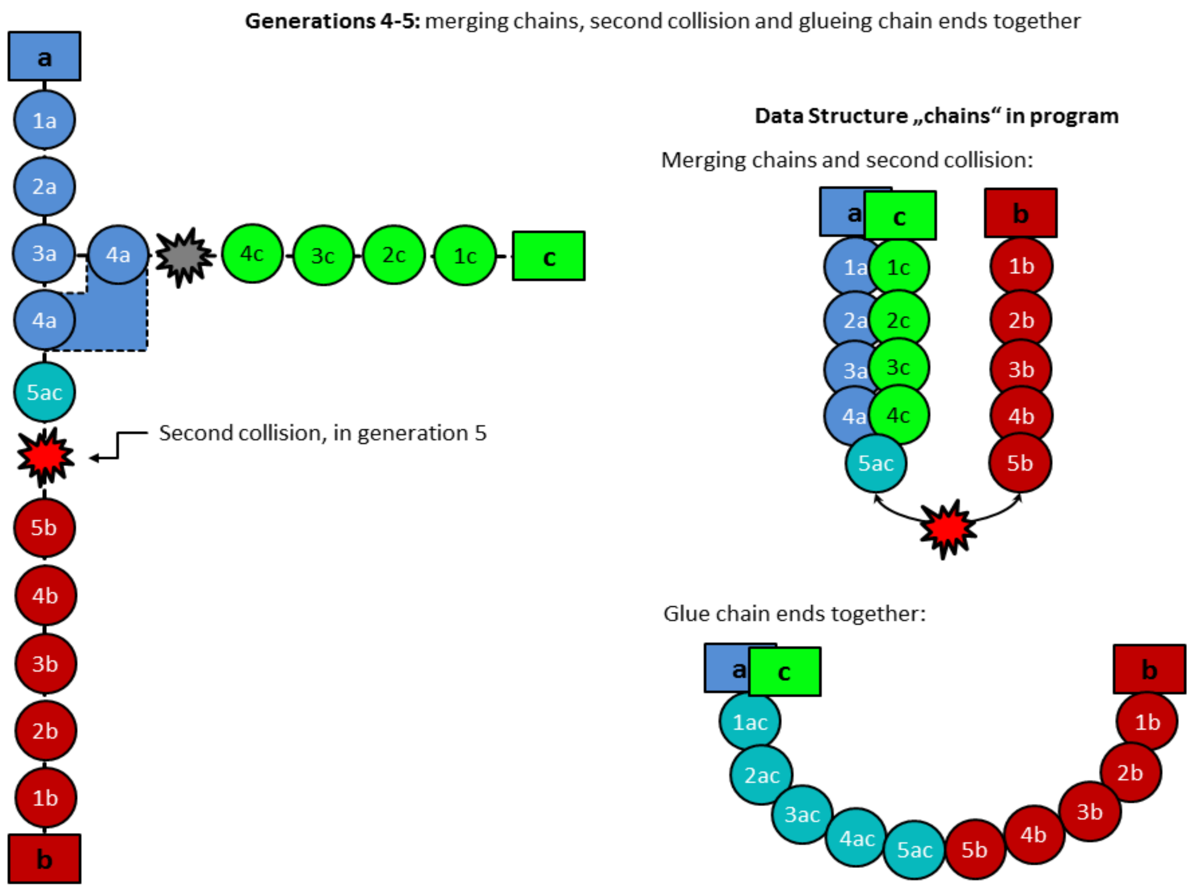

Generations 4-5: merging chains, second collision and glueing chain ends together
a
1a
2a
3a
4a
4a
4c
3c
2c
1c
c
5ac
Second collision, in generation 5
5b
4b
3b
2b
1b
b
Data Structure „chains" in program
Merging chains and second collision:
a
c
b
1a
1c
1b
2a
2c
2b
3a
3c
3b
4a
4c
4b
5ac
5b
Glue chain ends together:
a
c
b
1ac
2ac
3ac
4ac
5ac
5b
4b
3b
2b
1b

Figure 3: Partitioning of a T-junction molecule. Left side: Fully partitioned molecule. Right side, top: Data structure ”chains” in memory, after merging together chains a and c and continuing the partitioning to the last collision in generation 5. Two chains are merged by combining bins with the same indices. The two contact bins are also merged. The resulting chain will still fullfill the requirement that each bin have only two neighbours. Right side, bottom: Final chain formed by glueing together the ends of chains ac and b.

2.4 Merging dead ends

After the two last chains have collided, there still can be unsorted atoms. We will call these unsorted atoms ”dead ends”. To generate the desired data structure ”final chain”, these dead ends must be merged into the existing chains.

The merging process is shown in figure 4. Starting from the unsorted bin adjacent to the tip of the chain (here, bin eight), the atoms from each unsorted bin are assigned to the bins of the other chain, starting from the tip. This method ensures that the resulting final chain still fullfills the requirement that the atoms in each bin only interact with the atoms in at most two adjacent bins.

The dead end may be too long to be merged into the final chain. In this case, the dead end length can be halved by merging adjacent pairs of bins together. This process is repeated until the dead end is short enough to be merged into the final chain.

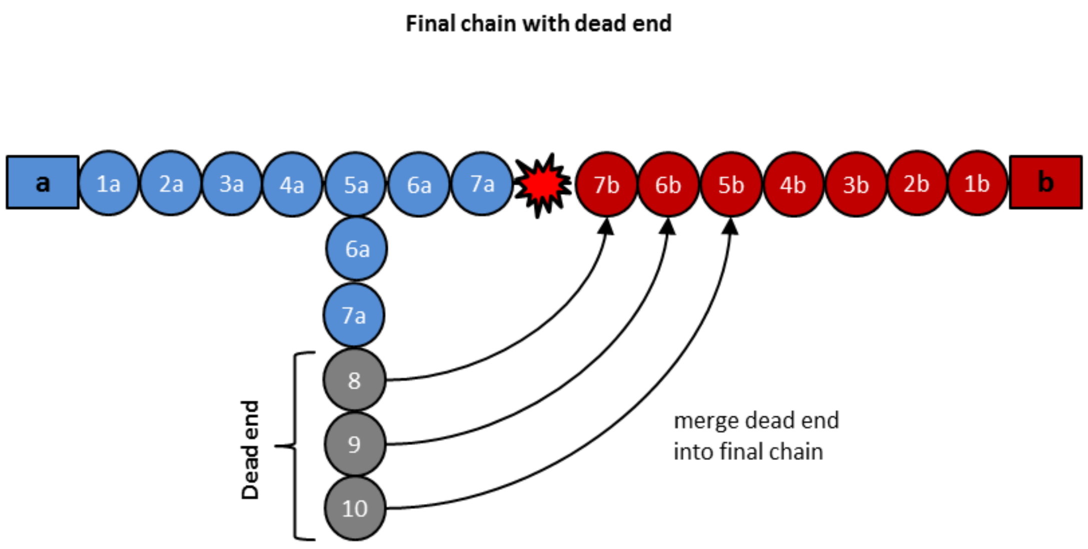


Figure 4: Merging dead ends. Dead end shown in dark gray.

3 Using the program

Before starting the program, the user needs to specify an input molecule geometry. The files defining the molecule are stored in the folder "input_files".

3.1 Input file format specification

Within the folder "input_files", there are two files defining each molecule, called "FILENAME.dat" and "FILENAME_transport.dat".

The "FILENAME.dat" file contains all the atom positions. In the first line, the number of atoms in the file is given. The second line is a comment line, which is skipped when reading the file. The following lines each specify the position of an atom in the molecule. Each of these lines starts with an element symbol, followed by three numbers, describing the X,Y and Z position of the molecule in a cartesian coordinate system. This file has the format of an ".XYZ" file, that can be displayed with molecule structure viewers such as Jmol.

The "FILENAME_transport.dat" file has two functions. First, it specifies which of the atoms in the molecule are part of the electrical contacts, and which are part of the device to be partitioned. Second, the file defines an interaction distance for each pair of element types.

The "FILENAME_transport.dat" file has the following format: The first line of the file starts with the keyword "Device" after which a range of device atom indices must be given, starting with index one. The indices correspond to the order in which the atoms are listed in "FILENAME.dat". The following lines define the contacts. They must start with the keywords "Contact1", "Contact2, and so on, followed once again by ranges of atom indices.

3.2 Starting the program

The entry point of the program is in "main.py". The program can be run using this file. Once the user has defined a molecule by writing two files "FILENAME.dat" and "FILENAME_transport.dat" to the folder "input_files", he must add an entry "FILENAME" to the list "ALL_FILE_NAMES" near the top of the file "main.py". The user can then specify which molecules are to be analyzed by splicing the appropriate parts of the list "ALL_FILE_NAMES" to the list "FILE_NAMES".

The program caches the martices (dist_mtrx) and (interact_mtrx). If the flag "LOAD_CACHE_DATA" is set to true, the program will load the cached matrices if the program is run run repeatedly. For large molecules, this can save considerable time.

The integer "MAX_GENERATIONS" defines the maximum number of generations of bins the program will generate during a run. This limit was defined to avoid accidental infinite loops. Its default value is 100, which may have to be increased for very large molecules.

Finally, if "GLOBAL_VERBOSITY_FLAG" is set to true, the program will print additional debugging information.