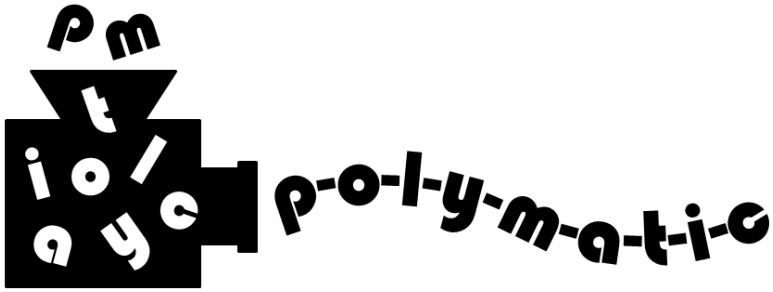
Polymatic User Manual

Version 1.0
March 18, 2013



The Colina Group
Department of Materials Science and Engineering
The Pennsylvania State University

http://www.matse.psu.edu/colinagroup/polymatic
https://nanohub.org/resources/17278

Contents

1 Introduction

1.1 About *Polymatic*

Polymatic is a code developed in the Colina Group for structure generation of amorphous polymers by a simulated polymerization algorithm. *Polymatic* can be thought of as a wrap-around code that calls a simulation package to perform energy minimization and molecular dynamics simulations. In between these simulations, the code performs polymerization steps by checking a set of defined bonding criteria and updating the connectivity information of the system (i.e., bonds, angles, dihedrals, impropers) appropriately to reflect the new bonds being made.

Polymatic is written to work with the Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS). Specifically, it is set up to read in and write out LAMMPS data files with a class II force field definition, including the 'class2' pair, bond, angle, dihedral, and improper styles. That being said, the majority of the subroutines included in the *Polymatic* code do not rely on these LAMMPS definitions, such that the code could be easily extended to work with other force fields, file types, and software packages.

1.2 Included Files

The distribution of *Polymatic* includes the following files:

1. `polymatic_manual.pdf` (this file), a basic user manual for *Polymatic*
2. `pack.pl`, a perl script that performs a random packing of molecules in a periodic cubic cell
3. `polym.pl`, a perl script that performs a single polymerization step
4. `polym_init.pl`, a perl script that performs an initialization of the system before polymerization
5. `polym_final.pl`, a perl script that performs a finalization of the system after polymerization

6. `polym_loop.sh`, a sample bash script to control the simulated polymerization procedure

7. `Example*/`, all files necessary to run the examples included in this manual

8. `COPYING`, a text file of the GNU General Public License

1.3 GNU Distribution Notice

Polymatic is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

Polymatic is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with *Polymatic*. If not, see `http://www.gnu.org/licenses/`.

1.4 Citation

The following paper describes the basic *Polymatic* algorithm and its implementation for a variety of linear polymers. If you publish work using the *Polymatic* code or variations of it, please cite this paper and the source code with the correct version specified.

Abbott, L. J.; Hart, K. E.; Colina, C. M. "Polymatic: A generalized simulated polymerization algorithm for amorphous polymers." *Theoretical Chemistry Accounts*, **2013**, *132*, 1334. DOI: 10.1007/s00214-013-1334-z

Abbott, L. J. *Polymatic: A simulated polymerization algorithm*, Version 1.0, **2013**, `https://nanohub.org/resources/17278`.

1.5 Publications Using Polymatic

Below is a growing list of published work using the *Polymatic* code or variations of it.

1. Abbott, L. J.; Colina, C. M. "Atomistic structure generation and gas adsorption simulations of microporous polymer networks." *Macromolecules*, **2011**, *44*, 4511–4519. DOI: 10.1021/ma200303p

2. Abbott, L. J.; Hart, K. E.; Colina, C. M. "Polymatic: A generalized simulated polymerization algorithm for amorphous polymers." *Theoretical Chemistry Accounts*, **2013**, *132*, 1334. DOI: 10.1007/s00214-013-1334-z

3. Hart, K. E.; Abbott, L. J.; Colina, C. M. "Analysis of force fields and BET theory for polymers of intrinsic microporosity." *Molecular Simulation*, **2013**. DOI: 10.1080/08927022.2012.733945

1.6 Acknowledgments

The development of *Polymatic* was supported in part by funding from the National Science Foundation (NSF) through grant DMR-0908781. Additionally, its development would not have been possible without computational resources and support provided by the Research Computing and Cyberinfrastructure unit of Penn State Information Technology Services and the Materials Simulation Center of the Materials Research Institute.

Thanks are also due to the people who have contributed to the development of *Polymatic*:

- Kyle E. Hart
- Grant Gonzalez
- Justin Hughes

2 Random Packing

2.1 About

The random packing code included with this distribution (`pack.pl`) packs any number of reference molecules into a periodic cubic simulation cell. A specified number of each reference molecule is packed into the periodic cell one at a time, where insertions are made avoiding only hard-core overlaps of the van der Waals radii. For ease of use with the *Polymatic* simulated polymerization code, the packing code was written to read in and write out LAMMPS data files.

Users should note that the intention of the packing code is to provide initial systems of molecules and/or polymeric repeat units at low densities, such as for use with the *Polymatic* simulated polymerization code. It was not designed to handle packing of large and complex molecules at high densities. Nor does it implement any type of Monte Carlo moves to introduce flexibility into the original reference molecule structures, nor any restraints on regions in which to pack molecules.

While the random packing code is included in this distribution, it is not required to generate initial structures for the *Polymatic* simulated polymerization. Initial structures can be obtained in any way, including use of other codes and software packages. Some examples of other software that can perform packing tasks include:

1. Amorphous Cell in the Materials Studio commercial software package, which implements a Monte Carlo packing algorithm.
   http://accelrys.com/products/datasheets/amorphous-cell.pdf

2. Packmol, an open-source software for packing molecules in defined regions of space, including possible spatial constraints.
   http://www.ime.unicamp.br/~martinez/packmol

3. Monte Carlo for Complex Chemical Systems (MCCCS) Towhee, which includes several Monte Carlo algorithms such as configurational-bias.
   http://towhee.sourceforge.net

2.2 Syntax

The syntax of the random packing code is:

```
./pack.pl -i num F1.lmps N1 F2.lmps N2 (...)
          -l boxL
          -o pack.lmps
```

Below is a description of all the command-line flags for this program. Note that the order in which the flags are specified is not important, but the order of the variables included for each individual flag is.

1. The `-i` flag indicates the reference molecule files (input files) and the number of each to be packed in the periodic cubic cell. Any number of reference molecules can be read in, which is indicated by `num`, but each molecule must be provided in its own data file, specified as `F1.lmps`, `F2.lmps`, etc. The number of molecules of each reference molecule to pack is given by `N1`, `N2`, etc.

2. The `-l` flag is used to define the length of the sides of the cubic simulation cell.

3. The `-o` flag gives the name of the output LAMMPS data file to be created, which contains the packed molecules.

4. The `-h` flag prints the syntax, then quits the program.

Note that if more than one reference molecule is provided, all molecules of reference type 1 (`F1.lmps`) are packed first, followed by the molecules of reference type 2 (`F2.lmps`), and so on. For most efficient performance, it is advised that the larger molecule types are packed first.

2.3 Output

The primary output of the packing code is the LAMMPS data file containing the packed system (`pack.lmps`). Additionally, output is printed to the command prompt showing the progress of the packing. When reading in a new reference molecule, a header line is printed showing the number and file name

of the reference molecule. Then, after each successfully packed molecule, the molecule number is printed, followed by the number of insertions attempted for that molecule.

For example, for a packing of 5 molecules of one reference type, output similar to the following would be generated:

```
Packing molecule type 1: F1.lmps
  1, 1 attempts
  2, 2 attempts
  3, 1 attempts
  4, 6 attempts
  5, 3 attempts
```

For a packing of 3 molecules of one reference type and 2 molecules of a second reference type, output similar to the following would be generated:

```
Packing molecule type 1: F1.lmps
  1, 1 attempts
  2, 1 attempts
  3, 4 attempts

Packing molecule type 2: F2.lmps
  4, 2 attempts
  5, 5 attempts
```

2.4 Notes

Below is a list of notes about the packing code that may be useful for using and/or modifying it.

1. When a LAMMPS data file of a reference molecule is read in, all information (atoms, bonding, coefficients, etc.) is stored in variables and arrays proceeded by `ref` (i.e., `$refNumAtoms` and `@refAtomType`). Likewise, the packed system information is stored in a similar set of variables and arrays proceeded by `pack` (i.e., `$packNumAtoms` and `@packAtomType`).

2. A maximum number of insertion attempts for each molecule is defined to prevent the code from running for infinite time. The default value is 1,000,000, but can be changed if desired. This variable is defined as `$maxAttempts`.

3. A molecule is inserted in the box through a random translation and rotation. The reference molecule is centered at the origin so that the rotation is made about this point. As currently written, the geometric center is used, but the code could be easily modified to make this the center of mass by incorporating the masses of each atom. The molecule centering takes place in the `readRef` subroutine.

4. Insertions of molecules are made to avoid overlap of the van der Waals radii of every intermolecular atom pair. The overlap is tested between the atoms based on the sigma values defined in the Nonbond Coeffs section of the LAMMPS data file. This overlap criterion is set in the `checkOverlap` subroutine.

5. To improve the efficiency of the code, neighbor lists are employed during the check for overlaps, which are updated with the addition of a new molecule. The width of the bins is set by the largest sigma value defined in the Nonbond Coeffs section of the LAMMPS data file. The neighbor list definitions are made in the `addMol` subroutine.

2.5 Examples

Two examples are provided here for using the random packing code. The first uses one reference molecule, the second uses two reference molecules. Illustrations of the molecules and final packed boxes are shown in Figure 1.

Example 1: This example will pack 45 repeat units of polystyrene into a periodic simulation cell with length 30 Å. The reference molecule file is `ps_monomer.lmps` and the packed system created is written to `pack.lmps`. The random packing is performed with the command:

```
./pack.pl -i 1 ps_monomer.lmps 45 -l 30 -o pack.lmps
```

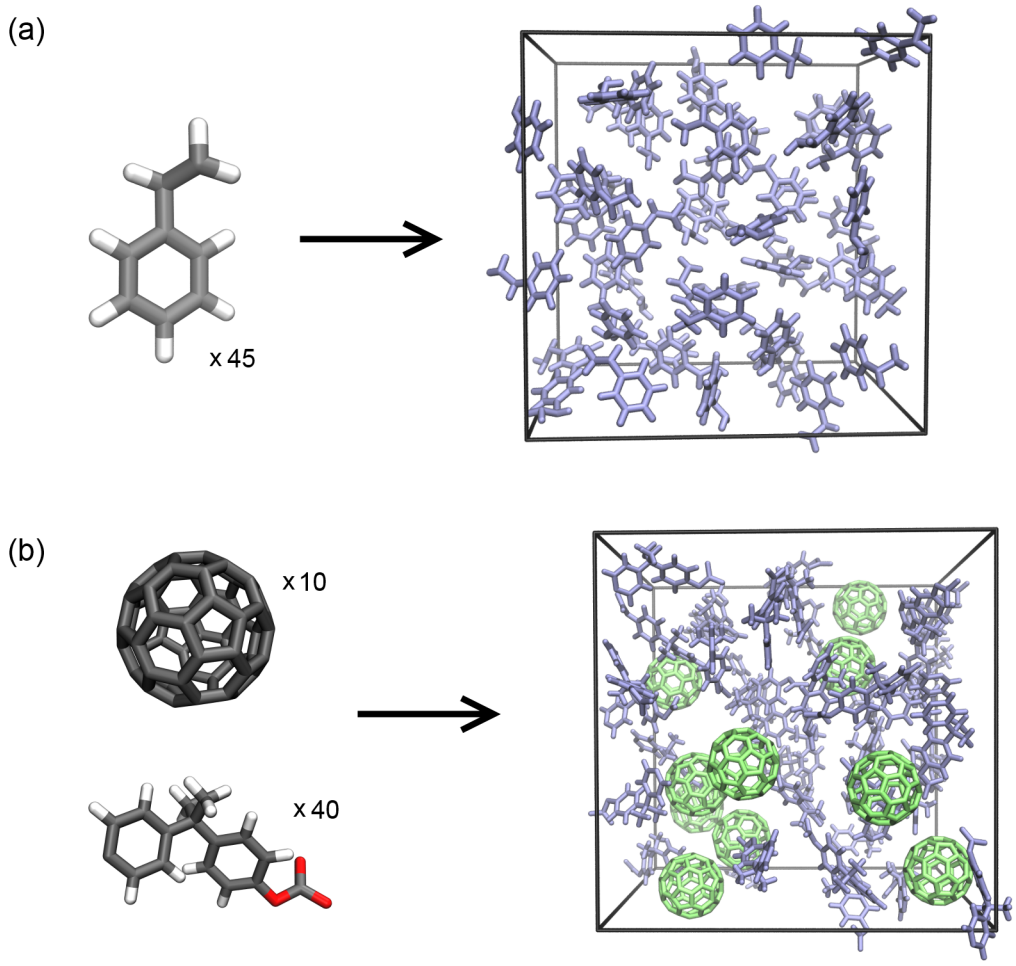


Figure 1. (a) A random packing of 45 repeat units of polystyrene in a periodic simulation cell of length 30 Å. (b) A random packing of 10 molecules of C_{60} fullerene and 40 repeat units of polycarbonate in a periodic simulation cell of length 40 Å. For clarity, the molecules of each reference type are shown in different colors in the packed systems.

Example 2: This example will pack 10 molecules of C_{60} fullerene and 40 repeat units of polycarbonate into a periodic simulation cell of length 40 Å. The reference molecule files are `c60.lmps` and `pc_monomer.lmps`, and the packed system created is written to `pack.lmps`. The random packing is performed with the command:

```
./pack.pl -i 2 c60.lmps 10 pc_monomer.lmps 40 -l 40 \
          -o pack.lmps
```

A special considerations should be noted when using more than one molecule, as in this example. The data types for all molecules must be consistent across the data files for each reference molecule, such that all data types are present in all files. For example, atom type 1 in the first reference molecule must be the same as atom type 1 in the second reference molecule, and so on.

Note: The input and output files for these two examples are provided with the *Polymatic* distribution in directories `Example1` and `Example2`.

3 Simulated Polymerization

3.1 About

The *Polymatic* distribution includes codes to perform a simulated polymerization of a simulation cell of monomers or repeat units. Using this approach, repeat units in close proximity are bonded according to a set of criteria, which take place along with energy minimization and molecular dynamics simulations in cycles to form a final polymeric system. A flowchart of the algorithm is given in Figure 2. It can be grouped into five main components:

1. *Polymerization initialization:* An initial system to be polymerized is provided, with the reactive atoms ("linkers") identified by unique atom types. Optionally, artificial charges can be added to linking atoms to encourage more efficient bonding in the polymerization process.

2. *Polymerization step:* The closest pair of linking atoms that satisfies all bonding criteria is selected, bond(s) are formed to obtain the proper polymeric structure, and an energy minimization is performed. If artificial charges were added, they are removed from the pair of bonded linker atoms. If no pair meeting all bonding criteria is found, a molecular dynamics (MD) simulation is performed and a polymerization step attempted again. Up to M_{max} polymerization steps are attempted, after which the polymerization loop (see below) is quit.

3. *Polymerization cycle:* N_{cyc} polymerization steps are performed. Then, an MD step is carried out to allow for relaxation and rearrangement of the structure. This composes a polymerization cycle. Multiple MD types can be defined and alternated throughout; in this algorithm, a second MD type is implemented every N_{md} cycles.

4. *Polymerization loop:* Polymerization cycles are repeated until B_{tot} bonds are formed or until no pair meeting the bonding criteria is identified in M_{max} attempts.

5. *Polymerization finalization:* The system is finalized to complete the polymerization. If artificial charges were added, they are removed from any remaining linker atoms.

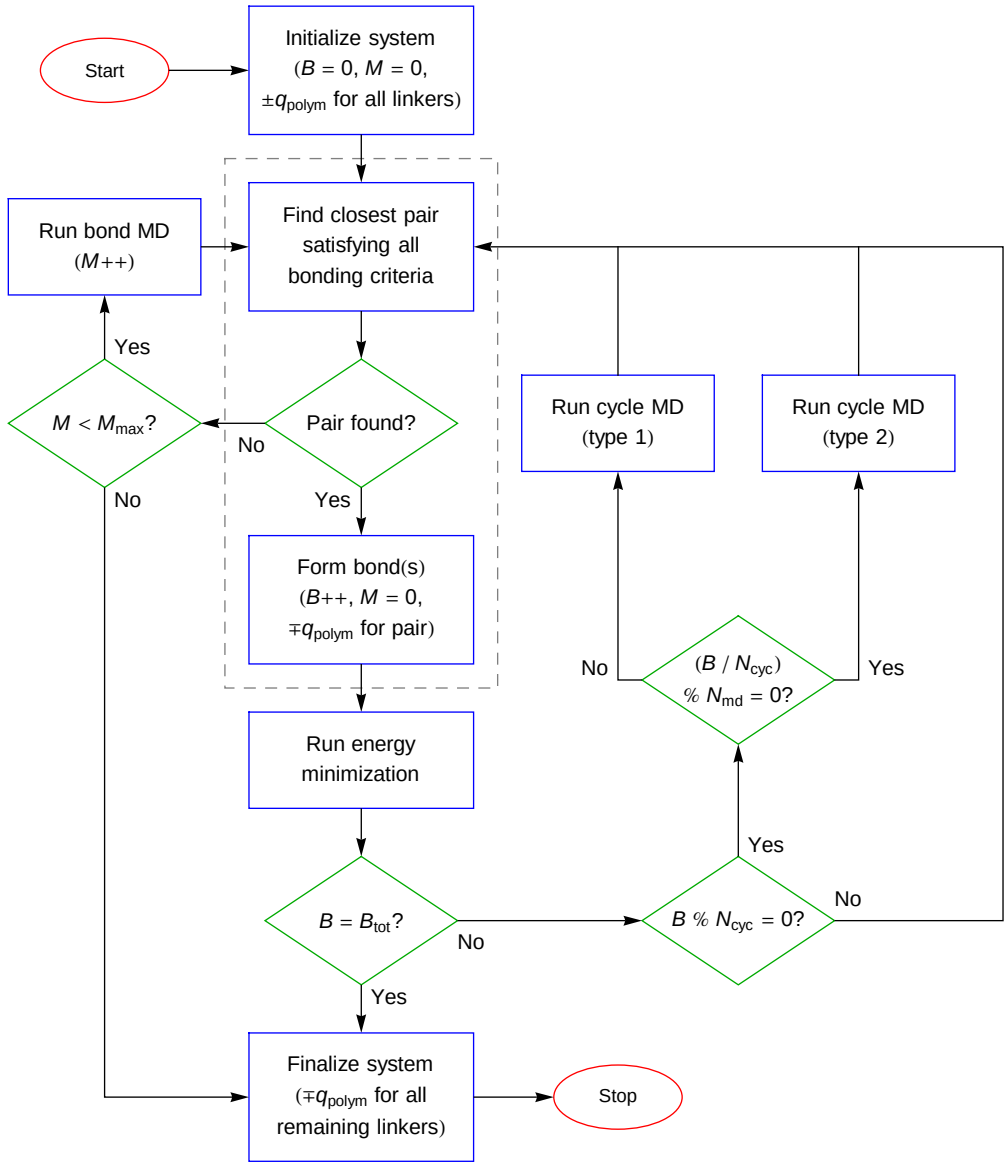


Figure 2. The flowchart of the simulated polymerization algorithm. The parts of the algorithm performed by the polymerization step perl code (`polym.pl`) are outlined in the dashed box. The remaining steps are controlled by the polymerization loop bash code (`polym_loop.sh`).

The algorithm is implemented using perl scripts to perform the initialization (`polym_init.sh`), a polymerization step (`polym_loop.sh`), and the finalization (`polym_final.sh`), as well as a bash script to control the polymerization loop (`polym_loop.sh`). These files will be described in the following sections. All codes described here were written to read in and write out LAMMPS data files, as well as to perform energy minimization and molecular dynamics simulations in LAMMPS. However, as many of the tasks performed in the algorithm are general, modifications could be made to work with other file types and software packages.

3.2 Polymerization Step

3.2.1 About

The polymerization step is an important part of the *Polymatic* algorithm, as it is responsible for identification of the closest pair of reactive atoms that satisfies all bonding criteria, as well as updating of the connectivity information of the system to reflect the new bond(s) being formed to produce the polymeric structure. The main tasks of the polymerization step, as outlined in a dashed box in Figure 2, are carried out in a perl script (`polym.pl`), which is described here.

3.2.2 Syntax

The syntax of the polymerization step code is:

```
./polym.pl -i data.lmps
           -t types.txt
           -s polym.in
           -o new.lmps
```

Below is a description of all the command-line flags for this program. Note that the order in which the flags are specified is not important.

1. The `-i` flag indicates the input data file of the initial system.

2. The `-t` flag indicates a types file, which identifies the data types in the LAMMPS data file. Its format is described in Section 3.2.3.

3. The `-s` flag indicates the input script, which specifies parameters for the polymerization step. Its format is described in Section 3.2.4.

4. The `-o` flag gives the name of the output data file to be created, which contains the system with updated connectivity. This file is not created if no pair of linker atoms is found meeting all bonding criteria.

5. The `-h` flag prints the syntax, then quits the program.

3.2.3 Types File

The types file (`types.txt`) is used to define the data types in the LAMMPS data file, such that the atom types involved in each bond, angle, dihedral, and improper are indicated. This is necessary to add the proper bonded interactions during the polymerization step after new bonds are formed. Additionally, the types file acts as a conversion key between the numerical atom types in a LAMMPS data file and more meaningful descriptive strings, such as those given by the force field.

The types file is specified in a text file, with a separate section for each data type, labeled 'atom types', 'bond types', 'angle types', 'dihedral types' and 'improper types'. In the atom types section, one line is provided for each atom type, where the numerical value from the LAMMPS data file is given first, followed by the associated string, space separated. The format of the atom types section is as follows:

```
atom types
 1 type1
 2 type2
 ...
```

After the atom types section, a section is listed for all other data types with a line for each numerical value, along with the atom types involved in that interaction. The atom types for each interaction should correspond to those given in the atom types section, and should be comma separated. The format

of the bond types and angle types sections is as follows (the dihedral types and improper types sections are similar):

```
bond types
 1 type1,type2
 2 type2,type6
 ...
angle types
 1 type1,type2,type3
 2 type4,type2,type7
 ...
```

Note that the order of the atom types in the definitions of the data types are not unique. For example, the bond, angle, and dihedral types are reversible, and can be provided in either order in the types file. For the improper types, the second type listed must be the central atom.

Examples of types files for three different polymers can be found in the example files included with the *Polymatic* distribution. These examples are described in Section 3.6.

3.2.4 Input Script

The input script (`polym.in`) is provided to pass important parameters for a polymerization step to the polymerization step code. Information given in the input script can include the linker atoms to be bonded, specifications for bonding criteria, optional artificial charges to add to linker atoms, additional bonds to be made other than between linker atoms, etc.

Each definition in the input script is given on a separate line, where the first word of the line is the command specifying which definition is provided. The commands currently implemented in *Polymatic* are listed below. The order of the command lines do not matter unless specified.

Linker atoms The linker atoms define the atoms between which bonds are to be formed during the polymerization steps. As specified by the *Polymatic* algorithm, a bond is formed between the pair of linker atoms in closest

proximity in the structure that meets all bonding criteria. Definition of the linker atoms is given in the input script by a `link` command as:

```
link  type1,type1_new type2,type2_new
```

Example:

```
link  Lc1,c1 Lc2,c2
```

Here, 'type1' and 'type2' are the atom types of the linker atoms, while 'type1_new' and 'type2_new' are new atom types assigned to the linker atoms after they are bonded. These types should be consistent with the strings defined in the types file.

Note that new atom types are given to bonded linker atoms to prevent them from participating in further polymerization steps. Therefore, unique atom types should be provided for linker atoms, which could be duplicates of other atom types in the system with a new name.

Artificial charges Artificial charges can be added to linker atoms to introduce stronger attractions between linker atoms during the molecular dynamics simulations. This can improve the efficiency of the polymerization by increasing the chances of two linker atoms coming in contact, as well as reduce the separation between linker atoms when bonds are formed. Definition of the charges is given in the input script by a `charge` command as:

```
charge  q1 q2
```

Example:

```
charge  +0.3 -0.3
```

Here, 'q1' and 'q2' are the charges added to all linker atoms during the polymerization initialization, corresponding to q_{polym} in Figure 2. Their order should be consistent with the linker atoms in the `link` command (i.e., 'q1' is added to 'type1' and 'q2' to 'type2'). After a bond is formed between two linker atoms, the charges are removed from that atom pair. During the polymerization finalization, artificial charges are removed from any remaining linker atoms at the end of the polymerization.

Note that the charges specified in the input script are added to the charges already present for the linker atoms, and do not replace their original charges. Adding charges of equal magnitude, one positive and one negative, maintains a charge neutral system.

Cutoff radius The cutoff radius is a *required* bonding criterion for the polymerization step. It specifies the maximum distance two linker atoms can be from one another in order to allow bond formation. If no pair of linker atoms is found within this distance, no bonds will be added during the polymerization step. Definition of the cutoff radius is given in the input script by a `cutoff` command as:

```
cutoff  r
```

Example:

```
cutoff  6.0
```

Intramolecular bonds A bonding criterion can be specified to allow or deny bonds to be formed within the same molecule. When polymerizing linear systems, for example, intramolecular bonds are generally prevented to keep the chains from forming loops. However, for network polymers, intramolecular bonding should be allowed. Definition of the intramolecular bonding criterion is given in the input script by an `intra` command as:

```
intra  true_false
```

Example:

```
intra  true
```

Note that the default value for intramolecular bonding is false.

Atom connectivity The remaining commands listed in this section involve atoms within the structure other than the linker atoms. In order to identify those atoms, the atom connectivity from the linker atoms to the other involved atoms is required. The linker atoms are assumed to be atoms

1 and 2 in this definition, in the same order they were defined in the `link` command. The connectivity and atom types sections (see below) should be provided at the end of the input script. Definition of the atom connectivity is given in the input script by a `connect` command as:

```
connect
1 connected_atoms_1
2 connected_atoms_2
3 connected_atoms_3
...
```

Here, the connectivity for each atom is given on a separate line, with the atom number given, followed by a comma-separated list of the atom numbers of the connected atoms. Only the connectivity to atoms referenced in other commands is required.

Note that the atom numbers given in the connectivity section are not the same as the atom numbers defined in the LAMMPS data file. They are uniquely defined for use only in the polymerization step to identify atoms within the repeat units.

A detailed example of a connectivity definition is provided here for ladder polymer PIM-1, two repeat units of which are illustrated in Figure 3a. In the figure, the linker atoms are marked by blue and red beads (atoms 1 and 2, respectively). The other atoms defined by atom numbers 3–12 are marked by green beads. The connectivity for this example is defined as:

```
connect
1 3,4
2 5
3 6
4 7
5 8,9
8 10,11
9 12
```

These extra atom definitions, atoms 3–12, are provided so the atoms can be referenced in other commands in the input script, examples of which are provided in the remainder of this section for PIM-1.

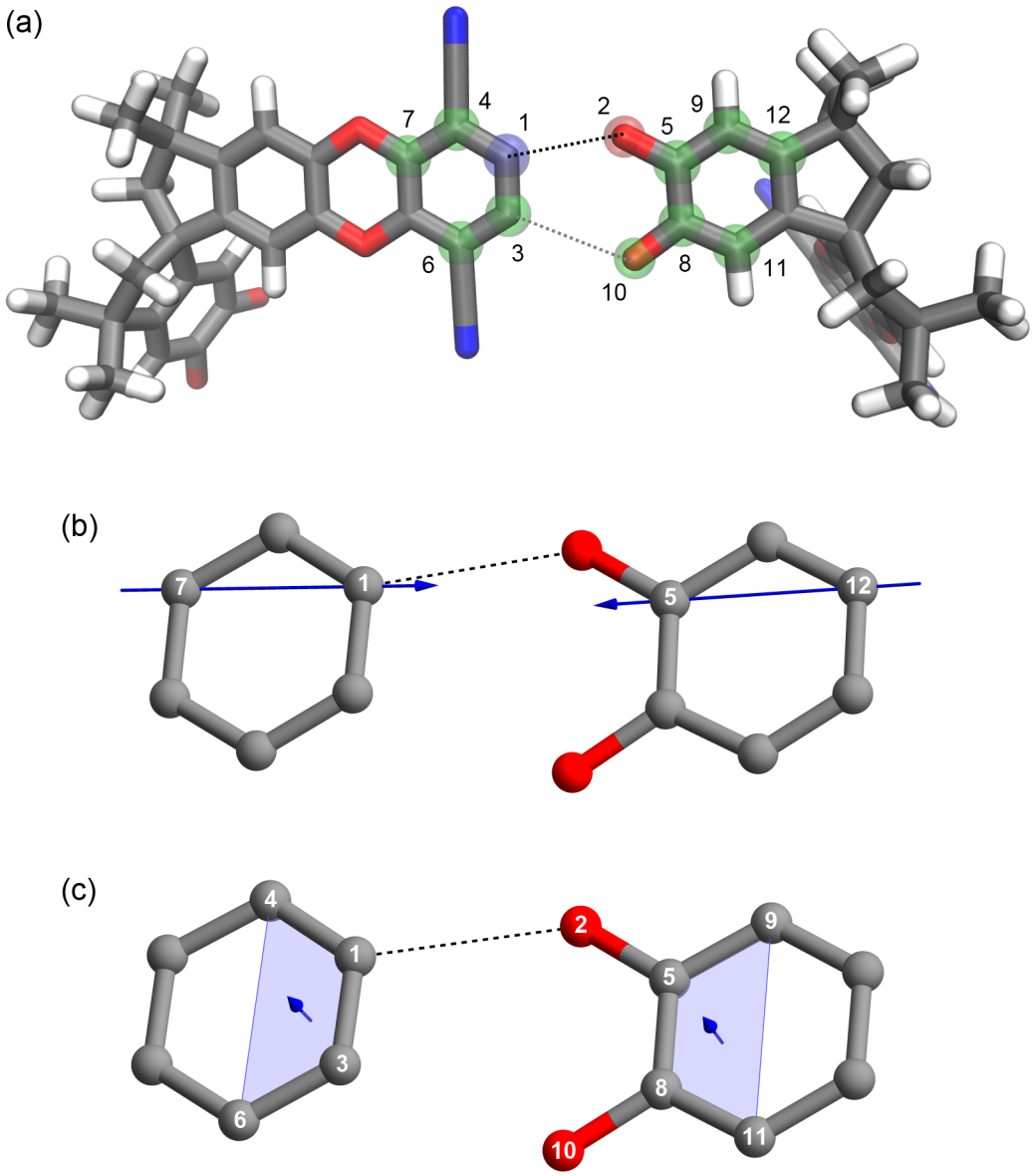


Figure 3. (a) Definition of a polymerization step for PIM-1, where the bonds to be formed are marked by dashed lines. Illustrations of (b) a vector check and (c) a plane check performed during the PIM-1 polymerization step, where the vectors and normal vectors are shown as blue arrows. The atom numbers correspond to the input script commands.

Atom types The atom types for the atoms specified in the connectivity section (see above) must also be given in order to identify the correct atoms within the structure. The atom types and connectivity sections should be at the end of the input script. Definition of the atom types is given in the input script by a `types` command as:

```
types
1 type1
2 type2
3 type3
...
```

Here, the atom type for each atom is given on a separate line, where the atom number corresponding to the connectivity section is given, followed by the atom type. The atom types should be expressed as strings consistent with the definitions given in the types file.

For the PIM-1 example, as illustrated in Figure 3a, the types section corresponding to the connectivity section defined above is given as:

```
types
1 Lcp
2 Loc
3 Lcp
4 cp
5 cp0
6 cp
7 cp
8 cp0
9 cp
10 Loc
11 cp
12 c5
```

Note that, in this example, an atom type 'cp0' was specified as a duplicate of 'cp' for atoms 5 and 8. This duplicate atom type was required to allow for a unique atom connectivity. This is discussed in more detail for the PIM-1 polymerization step example given in Section 3.6.

Extra bonds Polymerization steps can be defined for more complicated systems that require formation of an extra bond, in addition to the bond added between the closest linker atoms. Definition of an extra bond is given in the input script by a `bond` command as:

```
bond  atom1 atom2
```

Here, 'atom1' and 'atom2' correspond to the atom numbers given in the connectivity definition. Note that, if the atom types of the atoms involved in the extra bond are consistent with the linker atom types, then their atom types are changed after bond formation according to the `link` command, and artificial charges are removed if included. In this situation, the atoms in the extra bond should be specified in the same order as the `link` command.

For the PIM-1 example, shown in Figure 3a, the primary bond between the closest linker atoms is marked by the black dashed line, the additional bond by a gray dashed line. The extra bond for this example is given as:

```
bond  3 10
```

Vector checks Alignment checks can be imposed as bonding criteria to ensure that only reasonable bonds are formed during a polymerization step. Vectors can be defined by atoms within the system and the angle checked between those vectors to ensure it meets a specified set of criteria. Definition of a vector check is given in the input script by a `vector` command as:

```
vector  head1,tail1 head2,tail2 conditions
```

Here, the two vectors involved in the alignment check are specified by their head and tail atoms, identified by numbers as defined in the connectivity section. The conditions for the angle between the vectors are given last, which are specified as strict inequalities (i.e., $<$ or $>$).

For the PIM-1 example, vectors can be defined pointing out from each repeat unit, as illustrated by blue arrows in Figure 3b, the angle between which would have to satisfy $\theta > 135°$. This example is defined as:

```
vector  1,7 5,12 >135
```

Plane checks Alignment checks can be imposed also on the angle between normal vectors of best-fit-planes for atoms in the system. These work similarly to the vector checks in the previous section. Definition of a plane check is given in in the input script by a `plane` command as:

```
plane  atom1a,atom1b,... atom2a,atom2b,... conditions
```

Here, the two planes involved in the alignment check are specified by groups of atoms, identified by numbers as defined in the connectivity section. The conditions for the angle between the normal vectors of the planes are given last, which are specified as strict inequalities. If two separate conditions are required, they are joined by an AND (&) or OR (||) logical operator.

In the PIM-1 example, planes are defined by atoms in the aromatic rings to give normal vectors, as illustrated by the blue plane and arrows in Figure 3c, the angle between which would have to satisfy $\theta < 40°$ or $\theta > 140°$. This example is given as:

```
plane  4,1,3,6 9,5,8,11 <40,>140,||
```

Note that any number of vector and plane checks can be defined, each of which should be given as its own command in the input script.

3.2.5 Output

The primary output of the polymerization step code is a LAMMPS data file containing the system with updated connectivity (`new.lmps`). In addition, output is written to the command prompt specifying the distance between the linker atoms participating in the new bond formation (in angstroms), as well as the atom numbers of the atoms. If an extra bond is formed, a line is also printed to specify the atoms involved. For example:

```
Pair: 2.59 A (3,16)
Extra bond: (4,15)
```

Note that a new LAMMPS data file is written only if a successful polymerization step is completed. If no linker atoms are found within the system meeting all bonding criteria, no new file is written and an incomplete status code (3) is returned.

3.3 Initialization and Finalization

3.3.1 About

The initialization and finalization of the system are performed at the beginning and end of the polymerization loop, respectively, as indicated in the flowchart in Figure 2. Currently, the only task included in the initialization and finalization of *Polymatic* is the addition and removal of artificial charges to linker atoms, if specified by a `charge` command in the polymerization input script (see Section 3.2.4). In the initialization script (`polym_init.pl`), the artificial charges are added to all linker atoms within the system. Likewise, in the finalization script (`polym_final.pl`), the artificial charges are subtracted from any remaining linker atoms in the system.

If no artificial charges are used, the initialization and finalization steps need not be included in the simulated polymerization algorithm. Furthermore, these scripts could be updated to perform other tasks desired for initialization or finalization of the simulation, such as changing the atom types of all remaining linker atoms at the end of the simulation.

3.3.2 Syntax

The syntax of the polymerization initialization and finalization codes is:

```
./polym_x.pl -i data.lmps
             -t types.txt
             -s polym.in
             -o new.lmps
```

Here, `polym_x` should be either `polym_init` or `polym_final`.

Below is a description of all the command-line flags for this program. Note that the order in which the flags are specified is not important.

1. The `-i` flag indicates the input data file of the initial system.

2. The `-t` flag indicates a types file, which identifies the data types in the LAMMPS data file. Its format is described in Section 3.2.3.

3. The `-s` flag indicates the input script, which specifies parameters for the polymerization step. Its format is described in Section 3.2.4.

4. The `-o` flag gives the name of the output data file to be created, which contains the updated system.

5. The `-h` flag prints the syntax, then quits the program.

3.3.3 Output

The only output of the initialization and finalization scripts is a LAMMPS data file with the updated system. Currently, this includes only the addition or subtraction of artificial charges to all linker atoms within the system according to `charge` command in the input script (`polym.in`). If no charges are specified, no updates are performed, so that the output file is only a duplicate of the initial data file.

3.4 Polymerization Loop

3.4.1 About

The polymerization loop bash script (`polym_loop.sh`) controls the entire simulated polymerization algorithm, performing the appropriate tasks as illustrated in Figure 2. It is responsible for (i) calling LAMMPS to perform energy minimization and molecular dynamics simulations, (ii) calling the perl scripts described above to perform a polymerization step, initialization, or finalization, and (iii) handling file organization and maintenance.

3.4.2 Syntax

The polymerization loop takes no command-line arguments. All variables are defined in the beginning of the file or in input scripts. The syntax of the polymerization loop code is:

```
./polym_loop.sh
```

3.4.3 Variables and Input Scripts

The polymerization loop requires five variables, which are defined at the beginning of the script. They include:

1. `bonds`, the initial number of bonds formed in the polymerization, which corresponds to B in Figure 2. This variable should be 0, unless restarting the polymerization algorithm from the middle.

2. `bondsTotal`, the total number of bonds to be formed in the polymerization, which corresponds to B_{tot} in Figure 2. This number should not be changed if performing a restart.

3. `bondsCycle`, the number of bonds to form in each cycle, which corresponds to N_{cyc} in Figure 2. The cycle MD is performed when the logical statement $B \bmod N_{\text{cyc}} = 0$ is true.

4. `mdMax`, the maximum number of MD steps to perform during a bond attempt, which corresponds to M_{max} in Figure 2. The polymerization is quit if $M = M_{\text{max}}$ during a bond attempt.

5. `cycleMd`, the period of cycle MD type 2 simulations (in number of cycles), which corresponds to N_{md} in Figure 2. The type 2 MD simulation is performed when the logical statement $(B/N_{\text{cyc}}) \bmod N_{\text{md}} = 0$ is true.

All other parameters for the algorithm are provided in input scripts, the file paths for which are defined at the beginning of the bash script. These include a *Polymatic* input script (`polym.in`), which provides the parameters for the polymerization steps as discussed in Section 3.2.4, and LAMMPS input scripts, which give parameters for the energy minimization and molecular dynamics simulations.

3.4.4 Setup and Output

The main output of the polymerization loop are the files created during each step of the algorithm, including the files generated during LAMMPS molecular dynamics and energy minimization simulations. These are organized into directories during the polymerization, as described here.

The base directory should contain the initial LAMMPS data file and types file. There should also be a `scripts` directory that contains the polymerization step perl script, polymerization initialization and finalization perl scripts, *Polymatic* input script, and LAMMPS input scripts for the energy minimization and molecular dynamics simulations. Therefore, the initial setup should be:

```
polym_loop.sh      # Polymerization loop script
data.lmps          # LAMMPS data file of initial system
types.txt          # Types file
scripts/
  md0.in           # LAMMPS input script, bond MD
  md1.in           # LAMMPS input script, cycle MD type 1
  md2.in           # LAMMPS input script, cycle MD type 2
  min.in           # LAMMPS input script, minimization
  polym.in         # Polymatic input script
  polym.pl         # Polymerization step script
  polym_init.pl    # Polymerization initialization script
  polym_final.pl   # Polymerization finalization script
```

New directories and files are created as the polymerization progresses. First, the polymerization initialization is performed. A temporary data file is created, called `temp.lmps`, which always contains the most current system during the polymerization.

A directory is created for each bonding step, named `step_00N`. It includes three LAMMPS data files: (i) the initial system, `init.lmps`, which is copied from `temp.lmps` in the base directory, (ii) the updated system after a successful polymerization step, `data.lmps`, and (iii) the system after an energy minimization, `min.lmps`, which then replaces `temp.lmps` in the base directory. As such, each step will look similar to:

```
step_00N/
  init.lmps
  data.lmps
  min.lmps
```

When the polymerization step does not complete successfully, a molecular dynamics directory is created within the bond directory, named `md_M`, where

M corresponds to the MD step number, as defined in Figure 2. This directory contains two LAMMPS data files: (i) the initial system, `data.lmps`, which is copied from `init.lmps` in the bond directory, and (ii) the system after the molecular dynamics step, `md.lmps`, which then replaces `init.lmps` in the bond directory. Note that only one bond MD directory is kept; if additional steps are needed, the previous MD step is replaced with the current. A bonding step with molecular dynamics will look similar to:

```
step_00N/
  init.lmps
  data.lmps
  min.lmps
  md_M/
    data.lmps
    md.lmps
```

After the completion of a cycle, every N_{cyc} bonds, a cycle molecular dynamics step is performed in a new directory, named `step_00N_md1` or `step_00N_md2`, depending on whether it is the type 1 or type 2 cycle MD step. The directory contains two LAMMPS data files: (i) the initial system, `data.lmps`, which is copied from `temp.lmps` in the base directory, and (ii) the system after molecular dynamics, `md.lmps`, which then replaces `temp.lmps` in the base directory. As such, each MD step will look similar to:

```
step_00N_md*/
  data.lmps
  min.lmps
```

At the end, the polymerization finalization is performed. The final LAMMPS data file is written to `final.lmps`.

Note that output files generated by LAMMPS during energy minimization or molecular dynamics simulations are also written to their respective directories. In the provided bash script, all files are kept to allow tracking of the progress throughout, as well as debugging if issues arise. However, these files are not needed. The most current data file is always stored in the LAMMPS data file `temp.lmps`, such that any other temporary files can be deleted during or after the polymerization is completed.

In addition to the new files created during the polymerization, output is written to the command-prompt with the progress of the simulation. A header is written at the beginning with the input parameters. Then, a section is written for each bonding step, which includes the output generated by the polymerization step, as described in Section 3.2.5, as well as the number of bond attempts required for that step, defined by M in Figure 2.

For example, the output for a step during the polymerization might be:

```
Step 003:
  Pair: 3.18 A (36,123)
  Attempts: 6
```

In this example, the third bonding step completed successfully after six attempts, forming a bond between atoms 36 and 123, which were separated by a distance of 3.18 Å.

At the end of the polymerization, a footer is written to the output summarizing the number of bonds successfully created and the completion percentage of the polymerization.

3.5 Notes

Below is a list of notes about the polymerization codes that may be useful for using and/or modifying them:

1. It is important that all force field coefficients needed during the polymerization are provided in the original data file. This includes any new data types (atom, bond, angle, dihedral, and imporper) that are added to the system during a polymerization step. This can be observed, for instance, by comparison of the data files for the polystyrene repeat unit in Example 1 and the pair of repeat units in Example 3. If terms are missing, the polymerization step will quit with an error.

2. In the polymerization scripts, all system information (atoms, bonding, coefficients, etc.) is stored in variables and arrays (i.e., `$numAtoms` and `@atomType`). Generally, arrays of atom or bonding information is stored by the unique ID of the atom, bond, angle, etc.

3. The identification of the closest pair of linkers in the system is performed in the `findPair` subroutine. From there, all bonding criteria are checked for each pair, the checks for which can be performed in a separate subroutine (i.e., `alignment`). Any new bonding criteria implemented in *Polymatic*, should be included here.

4. After identification of the closest pair of linkers, all updates to the system information and connectivity are made in the `makeUpdates` subroutine. When a new bond is added, all possible angles, dihedrals, and impropers resulting from the new bonding are also automatically added to the system definition.

5. The *Polymatic* input script is read in the `readPolymInput` subroutine. If new commands for the input script are implemented, they should be added here.

6. Polymerization steps are implemented only for cubic simulation cells with periodic boundary conditions in all dimensions. Updated are required if other system setups are desired.

7. The way the algorithm is implemented in `polym_loop.sh`, it is possible to restart a polymerization from any bond step. To do so, the `bonds` variable in the script should be set to the number of the last bond completed, instead of 0.

8. The polymerization loop script (`polym_loop.sh`) is setup to run energy minimization and molecular dynamics simulations in LAMMPS using the setup for the High Performance Computing resources at Penn State. Modifications should be made to the lines in the script calling LAMMPS to be consistent with the setup being used, including the `energyMin` and `molDyn` functions, as well as the initial loading of the LAMMPS module at the end of the file paths section.

9. New data files are obtained after an energy minimization or molecular dynamics simulation in LAMMPS using the restart2data tool. It is not provided with the *Polymatic* distribution, but can be found with the additional tools in the LAMMPS distribution. Note that the version of the restart2data tool should be consistent with the version of LAMMPS being used for the simulation.

3.6 Examples

Three examples are provided here for using the polymerization codes. The first example (Example 3) performs a single polymerization step for polystyrene using the basic functionality of *Polymatic*. The second example (Example 4) performs a single polymerization step for PIM-1 with the more advanced functionalities. It also provides an instance in which a polymerization step is not successfully completed. The last example (Example 5) illustrates a full simulated polymerization of polycarbonate.

Example 3: This example will perform a single polymerization step on two nearby polystyrene repeat units, which are illustrated in Figure 4a. The closest pair of linker atoms is specified by blue and red beads connected by a dashed line, with atom types 'Lc1' and 'Lc2', respectively.

The original system is provided in `data.lmps`, where artificial charges have already been added to the linker atoms (± 0.3 e), and the types file in `types.txt`. The input script (`polym.in`) specifies the linker atoms, the artificial charges, and a cutoff radius of 6 Å. The polymerization step is performed with the command:

```
./polym.pl -i data.lmps -t types.txt -s polym.in -o new.lmps
```

On successful completion of the polymerization step, the following is output to the command prompt (`out`):

```
Pair: 3.20 A (23,8)
```

This signifies that a bond has been made between atoms 23 and 8 at a separation distance of 3.20 Å, as shown in Figure 4b. The atom types are changed to their new values ('c1' and 'c2', respectively) and artificial charges are removed from the pair.

Note that atom types 'Lc1' and 'Lc2' are specified as duplicates of 'c1' and 'c2' to provide unique types for use in the polymerization step. Changing the atoms types (e.g., from 'Lc1' to 'c1') after a successful polymerization attempt is necessary to ensure that these atoms do not participate in further polymerization steps.

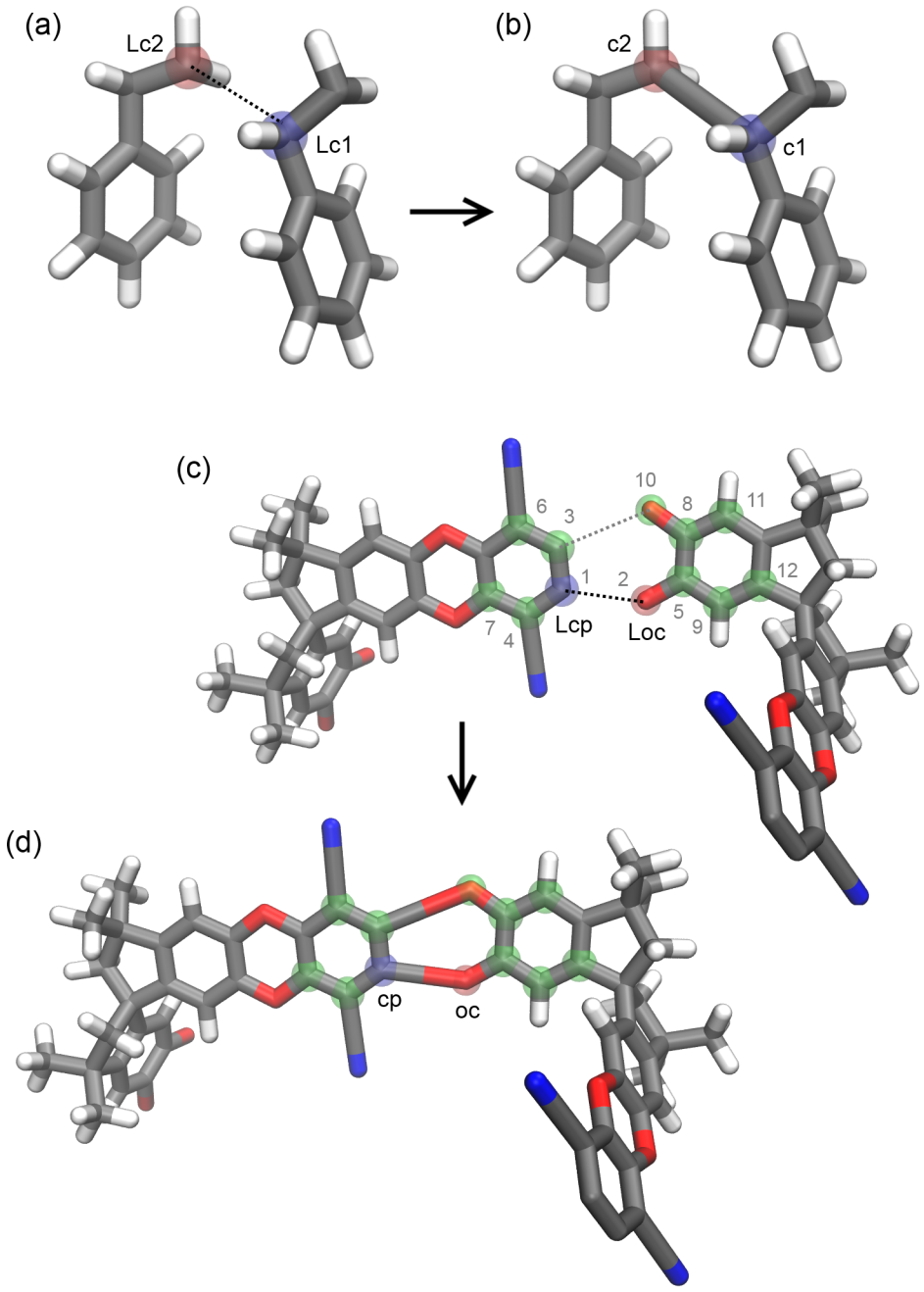


Figure 4. Illustration of two nearby repeat units of polystyrene and PIM-1 before (a,c) and after (b,d) a polymerization step.

A data file with the updated connectivity is also provided (`new.lmps`). Comparison of this file with the original system (`data.lmps`) indicates the new bond, angles, dihedrals, and impropers in their respective sections. The headers of the files provide the update counts of each:

```
# data.lmps          # new.lmps
32 atoms             32 atoms
32 bonds             33 bonds
48 angles            54 angles
64 dihedrals         79 dihedrals
16 impropers         22 impropers
```

Example 4: This example will perform a single polymerization step on two nearby PIM-1 repeat units, which are illustrated in Figure 4c. The closest pair of linker atoms is specified by blue and red beads connected by a black dashed line, with atom types 'Lcp' and 'Loc', respectively. Other atoms called in the input script commands are specified by green beads, and the additional bond required for the PIM-1 polymerization step is signified by the gray dashed line.

The original system is provided in `data.lmps`, where artificial charges have already been added to the linker atoms (± 0.3 e), and the types file in `types.txt`. The input script (`polym.in`) specifies the linker atoms, the artificial charges, a cutoff radius of 6 Å, the extra bond, the alignment checks, and the connectivity and atom types sections. The polymerization step is performed with the command:

```
./polym.pl -i data.lmps -t types.txt -s polym.in -o new.lmps
```

On successful completion of the polymerization step, the following is output to the command prompt (`out`):

```
Pair: 2.73 A (1,85)
Extra bond: (2,86)
```

This signifies that a bonds have been made between atoms 1 and 85 (separation distance of 2.73 Å) and between atoms 2 and 86, as shown in Figure 4d. Additionally, the atom types are changed to their new values ('cp' and 'oc', respectively) and artificial charges are removed from the pair.

Note that atom types 'Lcp' and 'Loc' are specified as duplicates of 'cp' and 'oc' to provide unique types for use in the polymerization step. Changing the atoms types (e.g., from 'Lcp' to 'cp') after a successful polymerization attempt is necessary to ensure that these atoms do not participate in further polymerization steps.

Furthermore, atom type 'cp0' is defined as a duplicate of 'cp' for atoms 5 and 8 in the atom types section of the input script to allow for a unique connectivity definition. Otherwise, when identifying the atoms bonded to atom 5, for example, it would be impossible to distinguish between atoms 8 and 9, which would typically both be of type 'cp'.

A data file with the updated connectivity is also provided (`new.lmps`). Comparison of this file with the original system (`data.lmps`) indicates the new bonds, angles, dihedrals, and impropers in their respective sections. The headers of the files provide the update counts of each:

```
# data.lmps          # new.lmps
110 atoms            110 atoms
120 bonds            122 bonds
216 angles           222 angles
346 dihedrals        361 dihedrals
104 impropers        106 impropers
```

Example 5: This example will perform the entire simulated polymerization algorithm on a periodic system of 40 polycarbonate repeat units, which are illustrated in Figure 5a.

In the base directory, the original system is provided in `data.lmps`, and the types file in `types.txt`. Additionally, the *Polymatic* input script, the LAMMPS input scripts, and the polymerization scripts are located in a directory called `scripts`. The input script (`polym.in`) specifies the linker atoms, the artificial charges, and a cutoff radius of 6 Å, while the LAMMPS input scripts specify the parameters for the energy minimization and molecular dynamics simulations.

The parameters of the polymerization are set in the 'User parameters' section of the `polym_loop.sh` script, which specify that 39 total bonds are to be

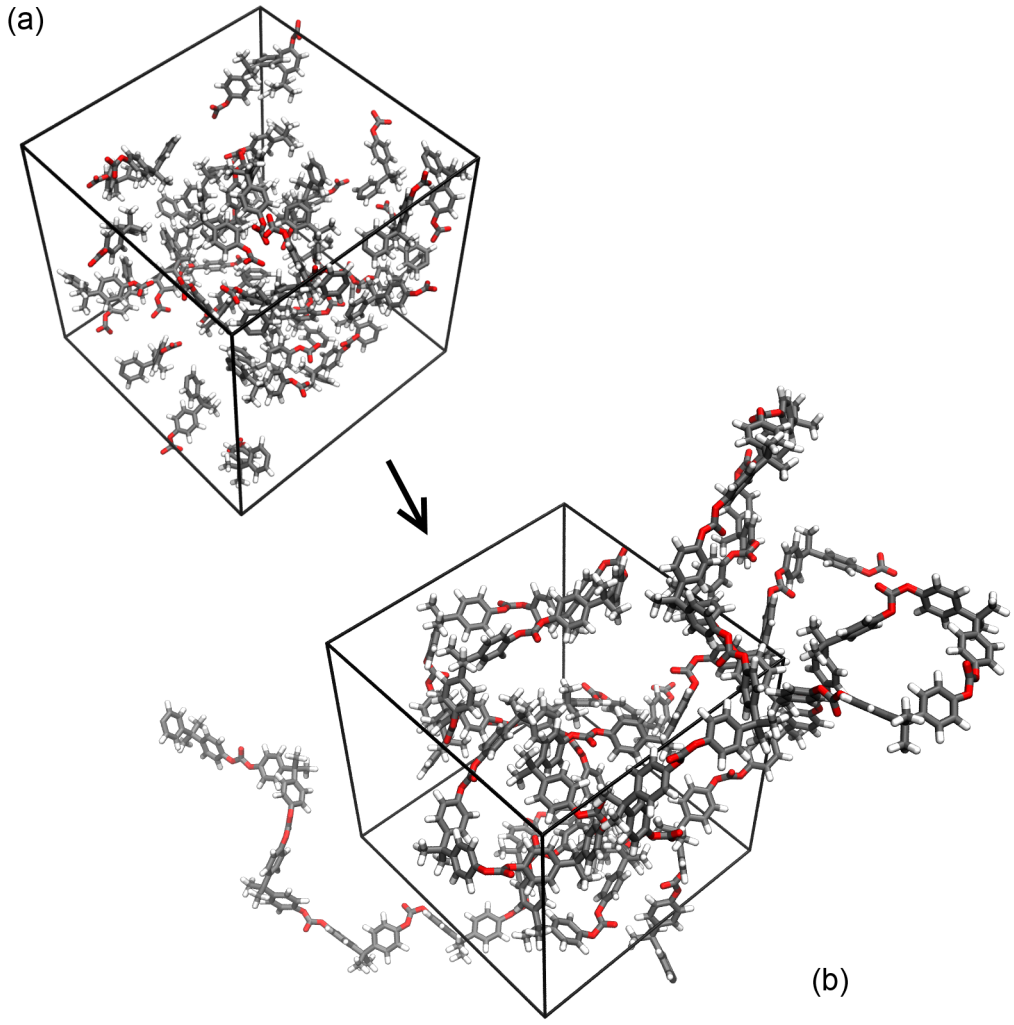


Figure 5. Illustration of a system of (a) 40 repeat units of polycarbonate and (b) the polymer chains after the *Polymatic* simulated polymerization is performed.

formed (`bondsTotal=39`), with cycles of 5 bonds (`bondsCycle=5`), a period of 3 cycles for the type 2 cycle MD simulations (`cycleMd=3`), and a maximum of 25 bond attempts (`mdMax=25`). File paths to all the necessary scripts are also given in the 'File paths' section of the script. The polymerization is performed with the command:

```
./polym_loop.sh
```

The final system of the polymerization is written to the LAMMPS data file `final.lmps`, which is illustrated in Figure 5b. Note that intermediate files are also generated during the polymerization, as discussed in Section 3.4.4, but are not included with the example files in the this distribution.

In addition to the new files generated, output is written to the command prompt summarizing the polymerization (`out`). This includes the parameters, the output for each polymerization step (as discussed in Section 3.2.5), and a summary of the run. In this example, 37 of 39 bonds were completed, resulting in a 95% completion.

Note: The input and output files for these three examples are provided with the *Polymatic* distribution in directories `Example3`, `Example4`, and `Example5`.