Aalborg Universitet

FragIt

A Tool to Prepare Input Files for Fragment Based Quantum Chemical Calculations

Steinmann, Casper; Ibsen, Mikael W.; Hansen, Anne S.; Jensen, Jan H.

[Link to publication from Aalborg University](#)

FragIt: A Tool to Prepare Input Files for Fragment Based Quantum Chemical Calculations

Casper Steinmann*, Mikael W. Ibsen, Anne S. Hansen, Jan H. Jensen

Department of Chemistry, University of Copenhagen, Copenhagen, Denmark

Abstract

Near linear scaling fragment based quantum chemical calculations are becoming increasingly popular for treating large systems with high accuracy and is an active field of research. However, it remains difficult to set up these calculations without expert knowledge. To facilitate the use of such methods, software tools need to be available to support these methods and help to set up reasonable input files which will lower the barrier of entry for usage by non-experts. Previous tools relies on specific annotations in structure files for automatic and successful fragmentation such as residues in PDB files. We present a *general* fragmentation methodology and accompanying tools called FragIt to help setup these calculations. FragIt uses the SMARTS language to locate chemically appropriate fragments in large structures and is applicable to fragmentation of any molecular system given suitable SMARTS patterns. We present SMARTS patterns of fragmentation for proteins, DNA and polysaccharides, specifically for D-galactopyranose for use in cyclodextrins. FragIt is used to prepare input files for the Fragment Molecular Orbital method in the GAMESS program package, but can be extended to other computational methods easily.

Introduction

The need to compute molecular properties for larger and larger systems with desirable accuracy has led to the development of novel methods such as fragmentation methods [1]. In fragmentation methods, a large system is divided into several smaller subsystems called fragments. Each fragment is treated with some *ab initio* level of theory and different methods [2–7] include the surrounding environment in different ways.

In this work, we are interested in setting up Fragment Molecular Orbital (FMO) [5,6] and Effective Fragment Molecular Orbital (EFMO) [8,9] calculations, but our method is extensible to other fragment based methods. In the FMO method, each fragment is polarized by the presence of the Coulomb field of all other fragments. The underlying equations allow for a systematic improvement of the energy by considering pairs and optionally triples of fragments [10], the latter often within milihartree accuracy of the corresponding *ab initio* energy. FMO supports correlated treatment of one or more fragments [11–13] as well the possibility of obtain excitation energies with good accuracy. [14] The FMO method in GAMESS [15] utilizes a novel parallelization scheme [16] to allow computations to be carried out efficiently on desktop computers as well as large scale super computers. [17] Fragmentation can occur across covalent bonds using either the Hybrid Orbital Projection (HOP) [18] or Adapted Frozen Orbital (AFO) [19,20] method. The EFMO method, also available in GAMESS, neglects the Coulomb bath from FMO and replaces it with classical terms to improve the computational speed. The input to EFMO and FMO are largely identical.

Often, the input files for the FMO method are more complex than the regular *ab initio* input files. The reason for this complexity is that complete knowledge about the individual fragments of interest are required, i.e. the atom indices that make up the fragment which might not be in any specific order in an input coordinate file, the integer fragment charges and level of theory. For a molecular cluster, each individual molecule can be considered a single fragment, for polymers a sub-unit of that polymer could make up a fragment whereas for proteins each individual residue can be considered a fragment. Fragmentation across covalent bonds adds more complexity: One must now also consider chemically reasonable places of fragmentation (do not break conjugation, etc.) which itself requires manual inspection of the structure of interest. Different systems have different complexities. For example, setting up a fragment calculation on a simple system consisting of three water molecules is feasible to prepare manually, but a protein with thousands of atoms is not. Consider also the case of multiple layers which the FMO method supports [21] in similar spirit to the ONIOM method [22,23] where one (or several) lower level layer(s) are used for some chemically irrelevant parts of a system but their effect on a higher level layer, which is used for the chemically interesting part, is needed. The assignment of fragments to individual layers usually based on the distance to a point of interest, is also no minor task when you have hundreds of fragments.

The need for automated tools which can setup calculations for a variety of systems (proteins, molecular clusters, polysaccharides, etc.) and automating the above tasks is thus of utmost importance
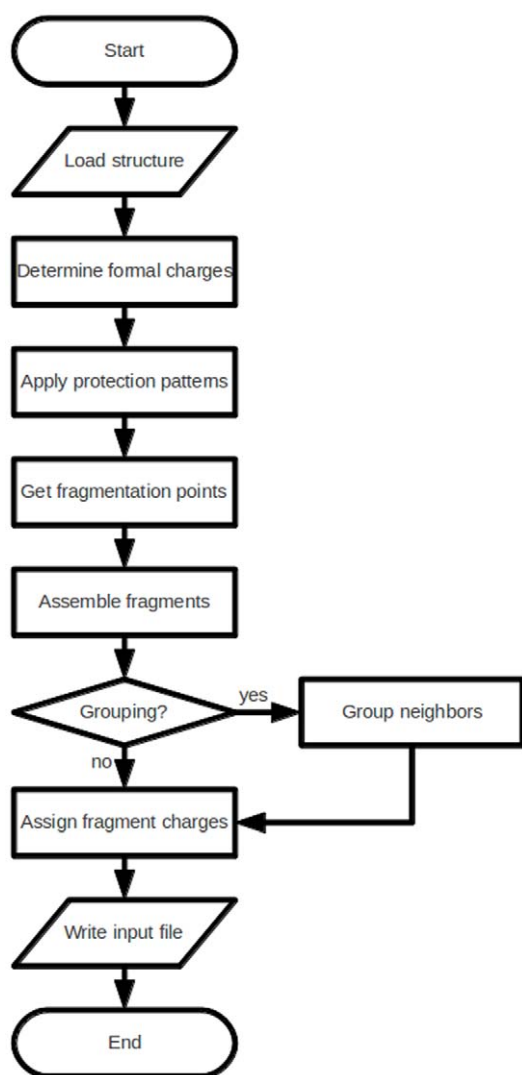
Figure 1. Scheme of the FragIt Algorithm.
doi:10.1371/journal.pone.0044480.g001
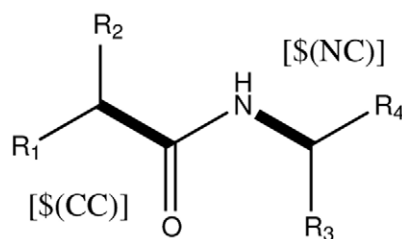
Figure 2. Pattern matching on a peptide bond using SMARTS.
doi:10.1371/journal.pone.0044480.g002

is supplied with the FMO method in GAMESS. It only supports fragmenting proteins/peptides and is dependent on a standardized PDB file format. It does include support for enabling multi-layered FMO calculations and letting the user choose whether to include solvation or not. Of more general applicability is the Facio [25,26] tool which supports the generation of FMO input files. The user can choose between fragmenting peptides, saccharides and nucleotides along with more specific options regarding the computational details such as level of theory and memory requirement. The main strength of Facio, however, is that it is a graphical user interface and one can define custom fragmentation bonds by using the mouse. While the FMOutil software is released under an open source license and can be run on any computer in a terminal, the Facio tool is closed source and available for Windows only.

We present an open source fragmentation methodology, an accompanying command line tool and a corresponding web service called FragIt that enables one to easily fragment any molecule or system of interest using predefined (or custom) patterns to locate fragmentation points. As output, FragIt creates an input file to the FMO method in GAMESS with reasonable defaults so the calculation can be started directly. FragIt can be extended to write input files for other (fragment) methods and new patterns of fragmentation can be created and tested without changing the source code. The only requirement for the input is that the structure of interest is protonated correctly in advance according to the problem of interest which can be achieved by tools such as PDB2PQR [27,28].

We have tested FragIt on several artificial and naturally occurring proteins with patterns of fragmentation to make reasonable fragments both in terms of the involved chemistry and size. We compare resulting fragmentation properties of FragIt and FMOutil and highlight similarities and differences. We also demonstrate that the fragmentation methodology is able to fragment a string of DNA and a polysaccharide successfully given

if these methods are to become viable tools that computational chemists use routinely in their research.

Software tools to prepare fragment method input files are already present but they differ greatly in their applicability and flexibility regarding different systems. The FMOutil [24] package

Table 1. Patterns of Fragmentation.

	Description	Match	LHS	RHS
Protein	fragmentation	$-C^{\alpha}$|$-C'-$	[$(CN)]	[$(C(=O)NCC(=O))]
	NH_2 protection	$-NH_2$	[$(NH2)]	CC(=O)[$(NCC=O)]
	NH_3^+ protection	$-NH_3^+$	[$(NH3)]	CC(=O)[$(NCC=O)]
Sugar	fragmentation	$-C$|$-C-$	[$(C1C(CO)OC(O)C(O)C1(O))]	[$(OC1C(O)C(O)CC(CO)O1)]
B-DNA	fragmentation	C|-C-	[$(CCOP)][$(CC1OCCC1)]	

Patterns used in FragIt for different types of chemical systems. The patterns for proteins include protection patterns to increase accuracy by eliminating very small fragments and the default fragmentation pattern makes sure to keep the quasi-conjugated nature of the peptide bond intact. For polysaccharides a single pattern to match α-D-galactopyranose units, the subunits of cyclodextrins, is included. We also include a pattern to fragment the backbone of DNA.
doi:10.1371/journal.pone.0044480.t001

the appropriate patterns. When SMARTS patterns are not possible, FragIt supports manual definition of fragmentation points. As an example, we fragment the Leucoemeraldine state of Polyaniline.

Design and Implementation

The FragIt algorithm was initially inspired by the RECAP [29] algorithm. The fragmentation algorithm (outlined in Figure 1 and discussed below) will take any file format supported by Open Babel [30,31] and fragment it, i.e. assign individual atoms to fragments, calculate the integer fragment charge from partial atomic charges in a fragment, locate atoms which define the boundaries of fragments and finally write an input file to the FMO method in GAMESS. The details of the implementation of the fragmentation algorithm is described below. The only required user input is a properly protonated and chemically reasonable structure.

To search for chemically reasonable places to fragment in molecules and to not rely on a single file format, we have based the fragmentation method on the SMiles ARbitrary Target Specification (SMARTS) [32] language which enables us to make substructure searches in molecules. For our specific needs, we have to find atomic species which are written in SMARTS as [M]. Here, M is an atomic primitive (an element such as a carbon atom). We are interested in locating pairs of atoms connected by a covalent bond through which we wish to define boundaries of fragments. A pair of atomic primitives is written as [M][N] where M and N are both atomic primitives. However, this is not nearly as flexible as we want since the chemical environment for different bonds vary, so to build atomic primitive environments we use the $() operator to define such environments as [$(MLHS)][$(MRHS)]. Here, MLHS and MRHS are general SMARTS match patterns for the left hand side and right hand side of a bond. The first atom in MLHS is covalently bound to the first atom in MRHS. For instance, to match atoms on each side of a peptide bond we might use [$(CC)][$(NC)] which would match a Carbon connected to another Carbon on one side to a Nitrogen connected to a Carbon on the other side (see Figure 2).

Due to the general application of SMARTS patterns, one can imagine protecting certain parts of a molecule from fragmentation is useful, say for a ligand in a protease or a specific residue in a protein. In FragIt, this protection is enforced via protection patterns which are SMARTS patterns to match atomic primitives as above.

FragIt is implemented in Python [33] and relies heavily on the use of the Open Babel API and its ability to be accessed through a SWIG [34] exposed Python interface [35].

The Fragmentation Algorithm

The fragmentation algorithm is outlined in Figure 1. Initially, the structure of interest is loaded via Open Babel and the partial charges of all atoms are obtained from the MMFF94 force-field via the OBChargeModel class. The MMFF94 force-field is specifically chosen in this work since it contains atom types aimed at systems of biochemical interest. However, Open Babel does include other force-fields which are probably more suited to inorganic systems. After having obtained the charges, potential atoms which should be protected by the appropriate SMARTS patterns (Table 1) are located. Hereafter follows the fragmentation procedure which fragments the system according to the fragmentation SMARTS patterns or explicitly defined valid pairs of atoms. In all cases SMARTS are handled by the OBSmartsPattern class from the Open Babel API and we obtain atom indices of fragmentation points directly because we base the search on atomic primitives.
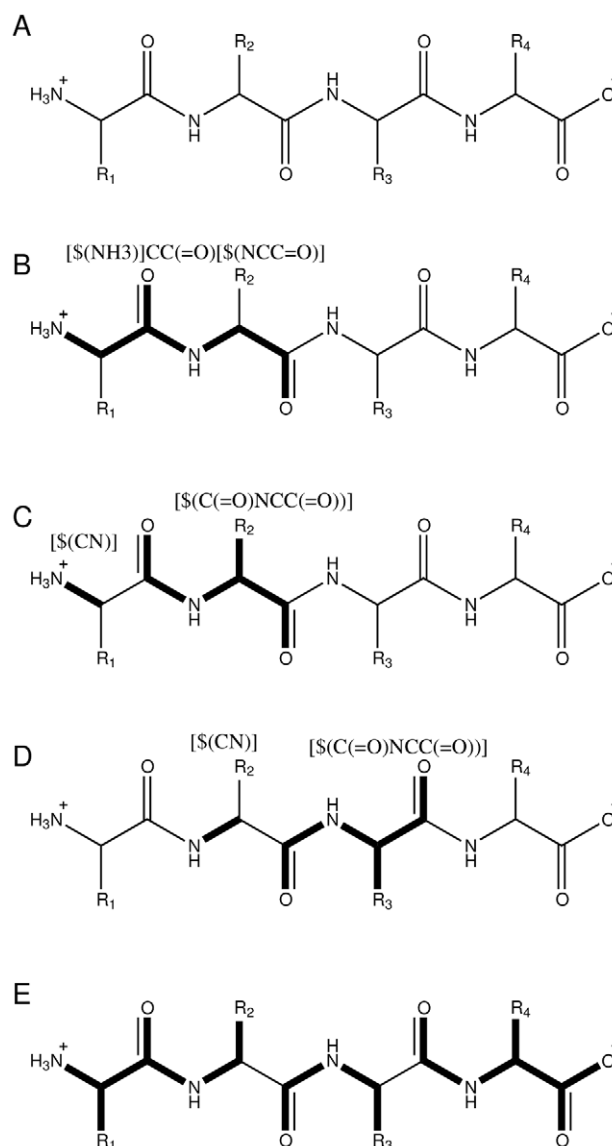
Figure 3. Protein Backbone Fragmentation Example. An example of how fragmentation and protection is carried out using SMARTS patterns on a protein in FragIt. Illustration a) is the uncorrupted protein backbone with side-chains R_1 through R_4, b) shows how a protection pattern matches atoms very specifically. c) and d) shows examples of fragmentation using the standard peptide pattern supplied in this work. Finally, e) is the final fragmentation when all fragmentation (4 fragments) and protection (1 fragment) is carried out, resulting in 3 fragments.
doi:10.1371/journal.pone.0044480.g003

To obtain the atoms that constitute a single fragment, we use the atom indices of the fragmentation points found above and with the FindChildren method of the OBMol class to extract all atoms between the two.

The fragmentation algorithm supports grouping neighboring fragments together. We have chosen an implementation which is a combination of two or more adjacent fragments into one. This combination of fragments has one benefit from a computational point of view, and that is to increase the accuracy of the computation.

Table 2. Default Fragmentation Results.

Protein	N_U	N_{frag}(1/2)	N_A^{max}(1/2)	Q	N_{S-S}	Comments
α-(ALA)$_{10}$	10	10/5	18/28	0	0	capped with methylene
α-(ALA)$_{20}$	20	20/10	18/28	0	0	capped with methylene
α-(ALA)$_{40}$	40	40/20	18/28	0	0	capped with methylene
β-(ALA)$_{10}$	10	10/5	18/28	0	0	capped with methylene
β-(ALA)$_{20}$	20	20/10	18/28	0	0	capped with methylene
β-(ALA)$_{40}$	40	40/20	18/28	0	0	capped with methylene
Chignolin	10	9/5	28/34	−2	0	
Tryptophan-cage	20	19/10	33/41	+1	0	
HPH	34	33/17	26/46	+1	0	
Crambine	46	42/23	28/43	0	3	
Glur2	259	264/138	34/46	+4	1	7 waters included
β-cyclodextrin	7	7/4	21/42	0	0	
B-DNA	24	26/14	33/64	0	0	
Leucoemeraldine	8	8/4	13/25	0	0	

Fragmentation results for some selected proteins, enzymes and sugar molecules. We show the number of units N_U in the molecules of interest, i.e. residues for proteins or sugars in the cyclodextrins, the number of resulting fragments N_{frag} both without (1) and with (2) grouping with the neighbor, the maximum number of atoms in a fragment both without (1) and with (2) grouping with the neighbor, the overall charge Q of the system at pH = 7 and the number of sulfur bridges N_{S-S} which in FragIt is automatically treated as one fragment.
doi:10.1371/journal.pone.0044480.t002

Once fragments are identified (and optionally grouped), we can assign fragment charges by counting the partial atomic charges in each fragment we obtained earlier.

Finally we write the input file for the FMO method in GAMESS.

Writing the Input Files

When the fragmentation data has been obtained an input file is written to disk for the FMO method in GAMESS. There are several options available when writing the input file. First, multi-layered input file generation is supported by selecting a fragment which is the *central fragment*. From this fragment the assignment to layers are calculated using the *minimum* distance R_{IJ} from fragment I, where I is the central fragment, to all other fragments J based on a user defined distance that separates one layer from another.

In addition to the above, support for the FMO Frozen Density (FMO/FD) [36] method is also included. In FMO/FD one has an active region in which we wish to do a geometry optimization, a buffer region to help relax the density of the active region and a frozen region where the density is kept frozen after initial convergence. One defines a central fragment as above and a distance. Fragments which have atoms within this distance from the central fragment should all be enabled for geometry optimization. Around this active region, a buffer zone is constructed similarly with a new distance within which atoms (and fragments) are considered a buffer. The rest is frozen.

The input file can be run directly in GAMESS because we provide sensible defaults which are not FMO specific (an energy calculation using RHF with a 3–21G basis set, 1 GB of memory per core, etc.). Specifically for FMO, only the AFO bonding scheme is supported since the original HOP formulation requires manual generation of molecular orbitals in the bonding region if one chooses exotic places to fragment. The HOP scheme is also dependent on an extra basis set input which is automatically done with AFO. While the defaults are only meant to get novice EFMO and FMO users started using fragment based methods we note

that manual changes to the input files for a production run is needed.

Finally, the ability to write an extra PyMOL [37] and/or Jmol [38] script to visually inspect the fragmentation (and optionally layering) is included. The scripts are based on templates which contains the markup needed to make the programs visually display the fragmentation. The fragment information from FragIt is then used to group atoms into fragments for visual display.

Figures in this paper were generated using the PyMOL functionality. The web service uses the Jmol functionality.

Description of the Test Dataset

To illustrate the applicability of FragIt to different molecules using different patterns, we show results for a molecular cluster consisting of water and a solute, Chignolin (PDB: 1UAO), Tryptophan-cage (PDB: 1L2Y), Human Parathyroid Hormone (HPH) residues 1–34 (PDB: 1ET1A), Crambine (PDB: 1CRN), GluR2 ligand binding core (PDB: 1FTJ) as well as several neutral methyl-capped α-helix and β-sheet alanine structures from Fedorov *et al.* [6] and finally a β-cyclodextrin (extracted from PDB: 3CGT). The neutral methyl-capped α-helices and β-sheets are good to show FragIt's capability to fragment and group fragments together. Chignolin, the Tryptophan-cage, Crambine and Glur2 are real world proteins and have charged termini. These will help show that FragIt can protect various parts of a protein (in this case $NH3^+$ groups) by grouping them with nearby fragments to increase the computational accuracy. Crambine and Glur2 also include Sulfur-bridges. The structure of B-DNA was obtain from Georgia State University [39]. Leucoemeraldine was built in Avogadro [40]. Lastly, the polysaccharide β-cyclodextrin is included to illustrate that by simply including the appropriate patterns of fragmentation these are fragmented as well.

The crystal structures of the proteins and β-cyclodextrin were protonated using PDB2PQR [27,28] at pH = 7.
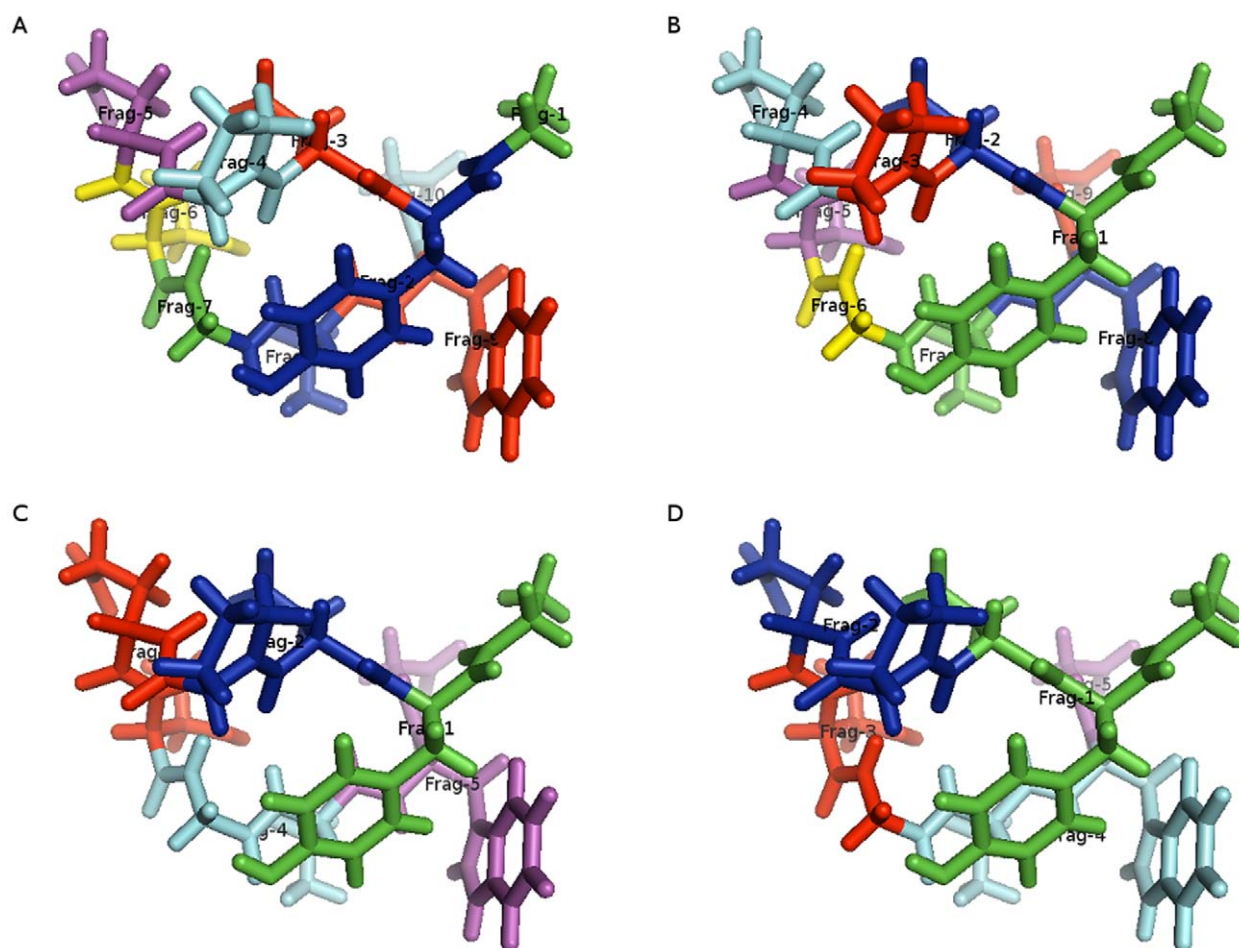
Figure 4. Different fragmentation options illustrated for Chignolin. Here shown a) without protection and no grouping, b) with protection and no grouping, c) without protection but in groups of two residues per fragment and d) with protection and using two residues per fragment. We use a six color coloring scheme resulting in different fragments may have the same color.
doi:10.1371/journal.pone.0044480.g004

Results

We now show four use cases of FragIt. First, we use it extensively to fragment proteins where we show many combinations of properties and compare with the FMOutil program. We then show that FragIt fragments polysaccharides, strands of DNA and other polymers as well. For an in-depth discussion of fragmentation patterns we refer to the Design and Implementation section. Groups of atoms not connected by a covalent bond are automatically grouped into separate fragments. So for example, any water molecules are treated as individual fragments.

Proteins

The fragmentation pattern used for peptide bonds is shown in Table 1 and illustrated in Figure 3[c–e]. Earlier work has shown [18,41] that fragmentation at the C^{α}-C' bond leads to higher accuracy energies for the FMO method compared to fragmentation at the semi-conjugated peptide bond. One has to also take care that fragments will not be too small and cause unphysical behavior. To facilitate this, we include several patterns of protection in FragIt. A protection pattern locates parts of a structure which must not be fragmented. The fragmentation pattern will generate small fragments at the N-termini (Figure 3c) but this fragment is chemically too small because of the positive

charge right next to a fragmentation point which will result in a too large inter-fragment charge-transfer. Furthermore, because of the way fragment boundaries are made using AFO uneven charges arising from the N-termini are also taken care of using protection patterns. The protection patterns (also listed in Table 1) will protect both a charged and a neutral N terminus. Protection of the C-terminus is implicitly built into the fragmentation pattern.

Shown in Table 2 are results for the several proteins. We list the number of residues for each protein as well as the number of fragments obtained after fragmentation. For the α-helices and β-sheets which are capped with methyl groups, we obtain as many fragments as residues. This is different from Chignolin, Tryptophan-cage and HPH where the protection patterns match the N-termini and (for one residue per fragment) make one less fragment than the number of residues. For GluR2, the results are similar, but there are also seven water molecules as well as two chains (which both get protected) yielding a total of 257 fragments in the protein plus 7 water fragments totaling 264. The column $N_{\mathrm{A}}^{\mathrm{max}}$ in Table 2 lists the maximum number of atoms in any fragment. For Crambine, the number of residues is 46 and the resulting number of fragments (for one residue per fragment) is 42 because one is protected and the three disulfide bridges are combined into three (rather than six) fragments. Disulfide bonds are not subject to automatic fragmentation in FragIt unless a specific pattern is

Table 3. Fragmentation of Chignolin using various options for FragIt and FMOutil.

N_{frag}	$N_A^{\min}$	$N_A^{\max}$	comment
FragIt			
9	7	28	default
10	7	24	no protection
6	14	28	default+merge
7	7	34	no protection+merge
5	10	40	group in pairs
5	21	34	group in pairs+no protection
FMOutil			
10	7	24	default
7	12	34	default+merge
5	21	34	group

doi:10.1371/journal.pone.0044480.t003

Figure 5. Sugar Fragmentation Example. An example of how fragmentation is carried using SMARTS patterns on a polysaccharide molecule in FragIt. Illustration a) is the polysaccharide molecule of interest, in this case an D-galactopyranose trimer. b) and c) shows examples of fragmentation using the polysaccharide fragmentation pattern in this work. Finally, d) shows the final fragmentation of 3 fragments.
doi:10.1371/journal.pone.0044480.g005

supplied. This is an intentional side-effect of using fragment based patterns because the S-S bond is comparatively very polarizable and therefore a poor choice for a covalent link.

Larger fragments leads to more accurate FMO results (see for example the work by Fedorov *et. al.* [42] for more information) and FragIt allows grouping of fragments where two covalently bonded fragments (such as two adjacent amino acids) can be grouped in to one fragment. This is illustrated in Figure 4 where fragments are colored to distinguish them. Because of using only 6 different colors, some fragments (although different) will have the same color. Here, a) shows Chignolin is fragmented without protection patterns using one residue per fragment. This results in 10 fragments with fragment sizes between 7 and 24 atoms (see Table 3) which is the same as the default behavior of FMOutil. b) shows the use of a protection pattern to prevent the first peptide bond from being cut. This is the default setting and gives 9 fragments with fragment sizes from 7 atoms to 28 atoms. We do note that the default setting gives slightly more imbalanced fragment sizes when thinking about parallelization strategy. To further improve fragment size balance one can choose to merge all Glycine residues with preceding fragments. This is a reasonable strategy to improve accuracy without added computational cost and is also possible in the FMOutil. With the default fragmentation behavior of FragIt and merging of Glycine residues the result is 6 fragments with fragment sizes ranging from 14 atoms to 28 atoms whereas not including protection patterns but merging gives rise to 7 fragments with fragment sizes from 7 atoms to 34 atoms. Correspondingly, merging Glycine residues with FMOutil yields 7 fragments with sizes 12 atoms to 34 atoms. The difference between FragIt and FMOutil lies in the way they see Glycine. FMOutil uses knowledge from residues to merge Glycine whereas FragIt relies on a pattern to find it. After fragmentation, the SMARTS pattern used does not recognize the N-terminal in Chignolin as a Glycine which gives rise to the discrepancy.

Including grouping but neglecting the protection pattern and merging gives rise to 5 fragments with a maximum of 37 atoms in a fragment (with the smallest fragment having 21 atoms). This is shown in Figure 4c) and the same fragment sizes that FMOutil does (Table 3). Compare this to d) where the combination of protection patterns and grouping leads to large size differences in fragments (minimum fragment size is 10 atoms and the maximum
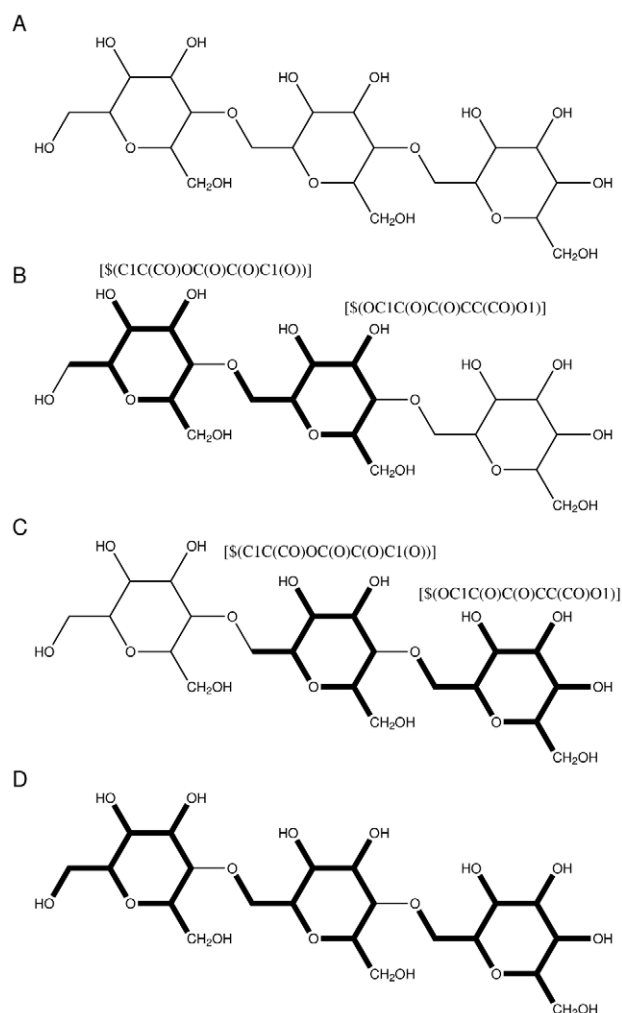
is 40 atoms). Different combinations of options can lead to very different fragmentation possibilities and care should be taken to fragment a system in the most sensible way in which the physics of the individual fragments is properly described.

Polysaccharides

Table 1 lists the fragmentation pattern we have included to fragment chains of D-galactopyranose. This pattern is specifically aimed at fragmenting cyclodextrins which are common in the design of artificial enzymes. The pattern explicitly matches the CH_2OH side chain. Figure 5[a] shows a trimer of D-galactopyranose and Figures 5[b–d] shows how the trimer is fragmented. The pattern takes an N-mer of sugar and converts it to N fragments, see Table 2. This also works for the cyclodextrins without any modifications. We did not observe any fragments during our tests that were of such size that protection patterns were necessary.

To fragment other sugars, one could modify the existing pattern to suit ones needs.
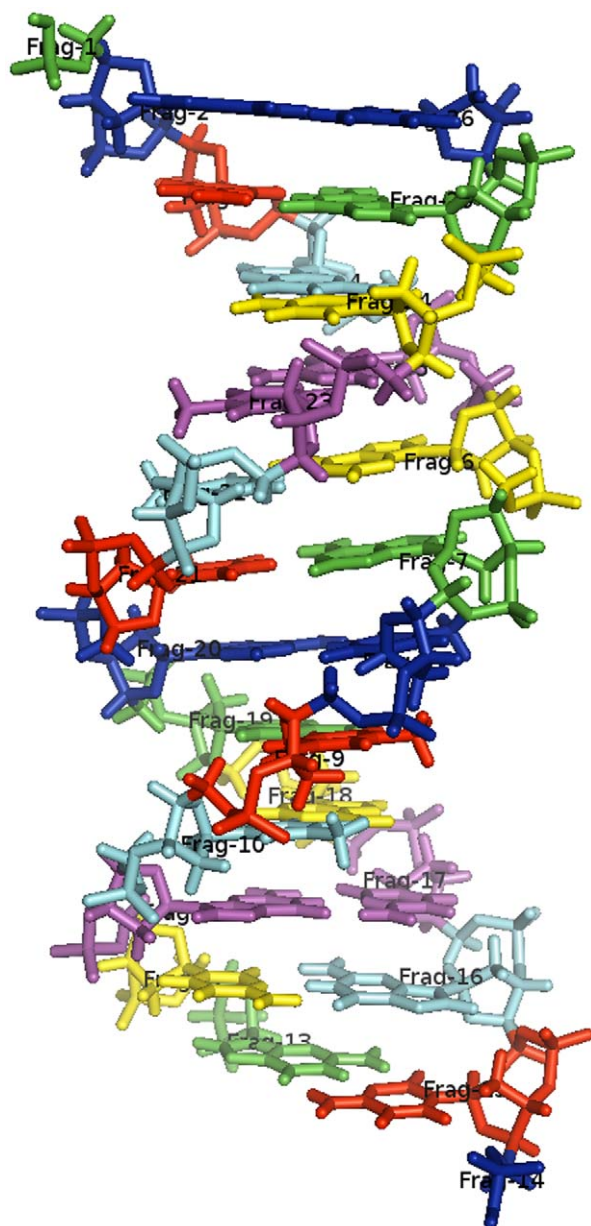
Figure 6. Fragmentation of DNA using FragIt. We use a six color coloring scheme resulting in different fragments may have the same color.

DNA

Although proteins has been the primary target of the majority of the fragment based methods, another interesting system to investigate is DNA. The nucleotides of DNA are connected to the DNA backbone which is made from sugars and phosphate groups connected by covalent bonds. We include a basic pattern (see Table 1) to fragment the backbone of DNA which can be observed in Figure 6. The 24 nucleotides were fragmented into 26 individual fragments and except from the two 9 atom fragments which make up the termini of each backbone (and could be protected by additional protection patterns) fragment sizes are 27 to 33 atoms depending on the type of nucleotide involved.

Other Polymers

Under some circumstances, a fragmentation pattern cannot be supplied, for instance when looking at Polyaniline since the right hand side and the left hand side of bond are the same which leads to multiple unwanted matches and very small fragments. To fragment such systems, it is possible to manually define pairs of atoms between which there should be a fragmented bond (see below). To show this feature, we have fragmented Leucoemeraldine, the fully reduced state of a chain of Alinine monomers (shown in Figure 7). This makes it possible to fragment systems even without having developed a fragmentation pattern beforehand. This option can be combined with regular fragmentation patterns allowing for very specialized fragmentation setups.

Another option which we might consider in the future is to implement a minimum fragment size option which could allow for these unwanted matches

Installation and Usage

The FragIt software is installed by downloading the source code from the project homepage (see below). FragIt requires Python v2.4 or later (not 3.X), Numpy [43] v1.5 (or later) and the Open Babel framework compiled with Python bindings version 2.3 (or later). Once these requirements are satisfied, FragIt can be executed from the command line. Options which are described in detail above is easily accessed through arguments to the command line executable. The web version requires Java to run Jmol, but otherwise no installation on a client computer.

To use FragIt invoke the executable and supply the structure which we wish to fragment (here Chignolin)

fragit 1uao.pdb

to generate an input file using the default settings (illustrated in Figure 4b). The default settings include fragmentation patterns for peptide bonds and sugar bonds as well as the listed protection patterns. To disable the use protection patterns use the –disable-protection command line option. All available options can be changed via command line options to the FragIt executable or via configuration files. To generate a configuration file named my.conf with the default fragmentation settings use the make-config option on a molecule of interest

fragit –make-config = my.conf

The contents of this file has all modifiable options and can be supplied to FragIt via the use-config option. To test new patterns
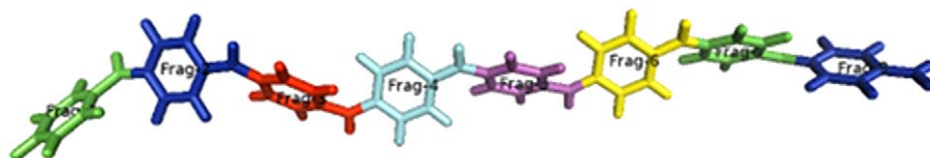

Figure 7. Fragmentation of Leucoemeraldine (a Polyaniline) using FragIt. This particular example uses no fragmentation patterns but rather explicitly defined points of fragmentation by the end user.
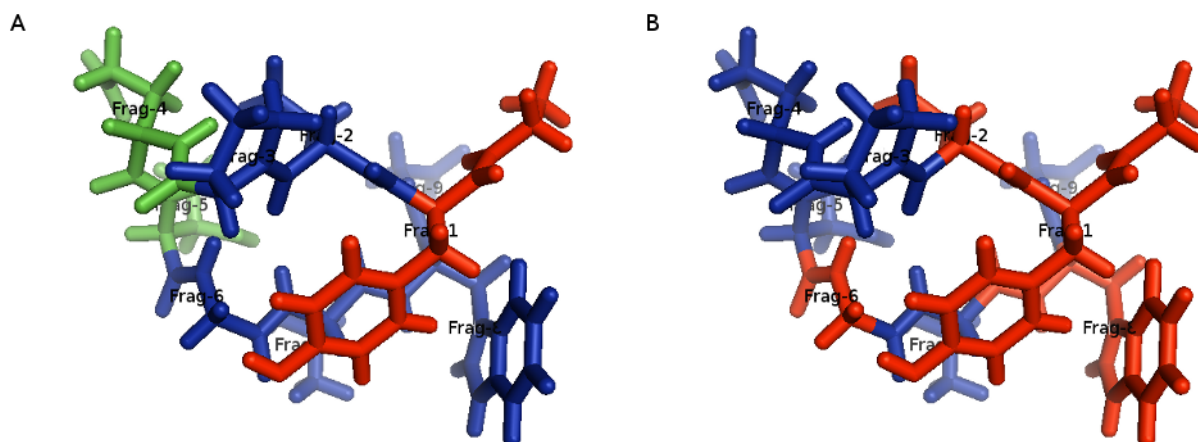
Figure 8. Different active, inactive and frozen regions in Chignolin. Regions are color-coded according to their function: red is active, blue is buffer and green is frozen. In a) fragment 1 is in the active region, fragments 2, 3, 6–9 are buffer region fragments and fragments 4 and 5 are in the frozen region. In b), fragments 1,2,6 and 8 are active fragments while fragments 3,4,5,7 and 9 are buffer region fragments. There are is no frozen region in b).
doi:10.1371/journal.pone.0044480.g008

for instance, this is the preferred approach since no tinkering with the FragIt source code is needed.

To invoke layered input files as described above, one must specify a central fragment (–output-central-fragment) as well as a distance (given by –output-boundaries) within which all fragments will be promoted to a higher level layer. This is accomplished by

fragit –output-central-fragment = 1 –output-boundaries = 3.0 1uao.pdb

which takes fragments within 3 Å of the first fragment and promotes it to a higher level layer.

To activate active, buffer and frozen regions, one must again specify both a central fragment and boundaries for higher level layers which are allowed to move [36]. Moreover, another distance within which atoms (and their fragments) are considered active is specified by the –output-active-distance option. Lastly, a buffer region of fragments between the active and the frozen is defined by a final distance (–output-buffer-distance)

fragit –output-central-fragment = 1 –output-boundaries = 3.0 \ –output-active-distance = 2.0 –output-buffer-distance = 3.0 1uao.pdb

which first will generate layers based on the boundary settings as above, then find active fragments within 2.0 Å from the central fragment, and finally create a buffer region around the active

Figure 9. Fragmentation of Chignolin using the web service.
doi:10.1371/journal.pone.0044480.g009

fragments within 3.0 Å from any active fragment. In this final step, some previously lower level layers may be promoted to higher level ones. The result is shown in Figure 8b. By changing the –output-active-distance option to 1.0, one obtains Figure 8a.

To group N consecutive fragments, one can use the –g N option. Usually, N would be 2 but FragIt supports any positive integer. To only merge Glycine the –merge-glycine option is available.

Lastly, to manually specify points of fragmentation one has to use configuration files and define the pairs string in explicit-fragmentpairs group. The format is pairs = A,B;C,D; where A, B, C and D are all atom indices and A,B is an atom pair between one wishes that there is an explicit bond.

Availability and Future Directions

The web service [44] available at www.fragit.org enables users to upload their structure, fragment it and download the resulting input file to GAMESS. The user is able to visually inspect (see Figure 9) the fragmentation using Jmol and make simple changes such as using multiple residues per fragment or enable layered input or optimization as discussed above. We are actively implementing features from the command line executable to work with the web service.

For greater flexibility, we strongly encourage the use of the command line tool which can be downloaded from the above URL or accessed from the development source at www.github.com/FragIt/.

The FragIt source code is distributed under an open source license (GPL, version 2 or later) and users of the FragIt code are encouraged to submit changes and additions, especially for their own (fragmentation) methods. It would also be possible to combine FragIt with other open source graphical tools such as Avogadro or even PyMOL, providing an alternative to the Facio software.

We plan on using and extending FragIt in the future with new patterns of fragmentation and methods as our research heads in new directions.

Author Contributions

Conceived and designed the experiments: CS JHJ. Performed the experiments: CS MWI ASH. Analyzed the data: CS MWI ASH. Contributed reagents/materials/analysis tools: CS MWI ASH. Wrote the paper: CS.

References

1. Gordon MS, Fedorov DG, Pruitt SR, Slipchenko LV (2012) Fragmentation Methods: A Route to Accurate Calculations on Large Systems. Chem Rev 112: 632–672.
2. Gao J (1997) Energy components of aqueous solution: insight from hybrid qm/mm simulations using a polarizable solvent model. J Comput Chem 18: 1061–1071.
3. Xie W, Gao J (2007) The design of a next generation force field: the x-pol potential. J Chem Theory Comput 3: 1890–1900.
4. Suárez E, Díaz N, Suárez D (2009) Thermochemical fragment energy method for biomolecules: Application to a collagen model peptide. J Chem Theory Comput 5: 1667–1679.
5. Fedorov D, Kitaura K (2007) Extending the power of quantum chemistry to large systems with the fragment molecular orbital method. J Phys Chem A 111: 6904–6914.
6. Fedorov D, Kitaura K (2009) The fragment molecular orbital method: practical applications to large molecular systems. CRC Press, Boca Raton, FL.
7. Mayhall NJ, Raghavachari K (2011) Molecules-in-molecules: An extrapolated fragment-based approach for accurate calculations on large molecules and materials. J Chem Theory Comput 7: 1336–1343. doi:10.1021/ct200033b.
8. Steinmann C, Fedorov D G, Jensen J H (2010) Effective fragment molecular orbital method: A merger of the effective fragment potential and fragment molecular orbital methods. J Phys Chem A 114: 8705–8712.
9. Steinmann C, Fedorov DG, Jensen JH (2012) The effective fragment molecular orbital method for fragments connected by covalent bonds. PLoS ONE 7: e41117. doi:10.1371/journal.pone.0041117.
10. Fedorov D, Kitaura K (2006) The three-body fragment molecular orbital method for accurate calculations of large systems. Chem Phys Lett 433: 182–187.
11. Fedorov D, Kitaura K (2004) Second order Møller-Plesset perturbation theory based upon the fragment molecular orbital method. J Chem Phys 121: 2483.
12. Fedorov D, Kitaura K (2005) Coupled-cluster theory based upon the fragment molecular-orbital method. J Chem Phys 123: 134103.
13. Fedorov D, Ishimura K, Ishida T, Kitaura K, Pulay P, et al. (2007) Accuracy of the three-body fragment molecular orbital method applied to Møller–Plesset perturbation theory. J Comput Chem 28: 1476–1484.
14. Chiba M, Fedorov D, Kitaura K (2007) Time-dependent density functional theory based upon the fragment molecular orbital method. J Chem Phys 127: 104–108.
15. Schmidt M, Baldridge K, Boatz J, Elbert S, Gordon M, et al. (1993) General atomic and molecular electronic structure system. J Comput Chem 14: 1347–1363.
16. Fedorov D, Olson R, Kitaura K, Gordon M, Koseki S (2004) A new hierarchical parallelization scheme: Generalized distributed data interface (gddi), and an application to the fragment molecular orbital method (fmo). J Comput Chem 25: 872–880.

17. Fletcher G, Fedorov D, Pruitt S, Windus T, Gordon M (2012) Large-scale mp2 calculations on the blue gene architecture using the fragment molecular orbital method. J Chem Theory Comput.
18. Nakano T, Kaminuma T, Sato T, Akiyama Y, Uebayasi M, et al. (2000) Fragment molecular orbital method: application to polypeptides. Chem Phys Lett 318: 614–618.
19. Fedorov D, Jensen J, Deka R, Kitaura K (2008) Covalent bond fragmentation suitable to describe solids in the fragment molecular orbital method. J Phys Chem A 112: 11808–11816.
20. Fedorov D, Avramov P, Jensen J, Kitaura K (2009) Analytic gradient for the adaptive frozen orbital bond detachment in the fragment molecular orbital method. Chem Phys Lett 477: 169–175.
21. Fedorov D, Ishida T, Kitaura K (2005) Multilayer formulation of the fragment molecular orbital method (fmo). J Phys Chem A 109: 2638–2646.
22. Svensson M, Humbel S, Froese R, Matsubara T, Sieber S, et al. (1996) Oniom: A multilayered integrated mo+ mm method for geometry optimizations and single point energy predictions. a test for diels-alder reactions and pt (p (t-bu) 3) 2+ h2 oxidative addition. J Phys Chem 100: 19357–19363.
23. Dapprich S, Komáromi I, Byun K, Morokuma K, Frisch M (1999) A new oniom implementation in gaussian98. part i. the calculation of energies, gradients, vibrational frequencies and electric field derivatives. J Mol Struc-Theochem 461: 1–21.
24. Fmoutil v. 2.1. URL http://staff.aist.go.jp/d.g.fedorov/fmo/fmoutil.html. Accessed 2012 August 1.
25. Suenaga M (2005) Facio: new computational chemistry environment for pc gamess. J Comput Chem Jpn 4: 25–32.
26. Suenaga M (2008) Development of gui for gamess/fmo calculation. J Comput Chem Jpn 7: 33–53.
27. Dolinsky T, Nielsen J, McCammon J, Baker N (2004) Pdb2pqr: an automated pipeline for the setup of poisson–boltzmann electrostatics calculations. Nucleic Acids Res 32: W665.
28. Dolinsky T, Czodrowski P, Li H, Nielsen J, Jensen J, et al. (2007) Pdb2pqr: expanding and upgrading automated preparation of biomolecular structures for molecular simulations. Nucleic Acids Res 35: W522.
29. Lewell X, Judd D, Watson S, Hann M (1998) Recap retrosynthetic combinatorial analysis procedure: a powerful new technique for identifying privileged molecular fragments with useful applications in combinatorial chemistry. J Chem Inf Comp Sci 38: 511–522.
30. O'Boyle N, Banck M, James C, Morley C, Vandermeersch T, et al. (2011) Open babel: An open chemical toolbox. J Cheminf 3: 33.
31. Openbabel v.2.3.0. URL http://openbabel.sourceforge.net. Accessed 2012 August 2.
32. James C, Weininger D (2006) Daylight theory manual, version 4.9. Aliso Viejo, CA, Daylight Chemical Information Systems Inc [Online] Available: http://www daylight com/dayhtml/doc/theory Accessed 2012 August 2.
33. Python. URL http://www.python.org. Accessed 2012 May 18.
34. Swig. URL http://www.swig.org. Accessed 2012 July 6.

35. Openbabel python. URL http://openbabel.sourceforge.net/wiki/Python. Accessed 2012 August 2.
36. Fedorov D, Alexeev Y, Kitaura K (2011) Geometry optimization of the active site of a large system with the fragment molecular orbital method. J Phys Chem Lett 2: 282–288.
37. DeLano W (2002) The PyMOL molecular graphics system. Schrødinger, LLC.
38. Jmol: an open-source java viewer for chemical structures in 3d. URL http://www.jmol.org. Accessed 2012 August 2.
39. Glactone, georgia state university. URL http://chemistry.gsu.edu/Glactone/PDB/pdb.html. Accessed 2012 August 2.
40. Avogadro v.1.0.3. URL http://avogadro.openmolecules.net. Accessed 2012 August 2.
41. Nakano T, Kaminuma T, Sato T, Fukuzawa K, Akiyama Y, et al. (2002) Fragment molecular orbital method: use of approximate electrostatic potential. Chem Phys Lett 351: 475–480.
42. Fedorov D, Kitaura K (2004) The importance of three-body terms in the fragment molecular orbital method. J Chem Phys 120: 6832.
43. Numpy. URL http://numpy.scipy.org. Accessed 2012 May 18.
44. Fragit web service. URL http://www.fragit.org. Accessed 2012 August 7.