QChASM: Quantum Chemistry Automation and Structure Manipulation

Victoria M. Ingman, Anthony J. Schaefer, Laura R. Andreola, and Steven E. Wheeler [*]

Article Type:

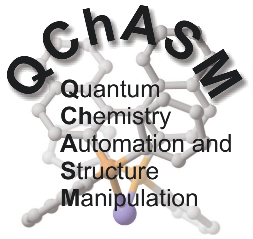
Software Focus

Abstract

As the tools of computational quantum chemistry have continued to mature, larger and more complex molecular systems have become amenable to computational study. However, studies of these complex systems often require the execution of enormous numbers of computations, which can be a tedious and error-prone process if done manually. We have developed a suite of free, open-source tools to facilitate the automation of quantum chemistry workflows. These tools are collected under the organization QChASM (Quantum Chemistry Automation and Structure Manipulation) and include functionality for building and manipulating complex molecular structures and performing routine tasks (AaronTools), a toolkit for automating TS optimizations and predictions of the outcomes of selective homogeneous catalytic reactions, and a plug-in for UCSF ChimeraX that provides a graphical interface for building complex molecular structures and representing output from quantum chemistry computations. These tools are described below, with a focus on the recent Python implementation of AaronTools.

[*]Department of Chemistry, University of Georgia, Athens, GA 30602; swheele2@uga.edu

GRAPHICAL TABLE OF CONTENTS



Caption: QChASM is a collection of free, open-source tools for building and manipulating complex molecular structures and automating quantum chemistry applications.

INTRODUCTION

A growing challenge in modern applications of computational quantum chemistry is managing the sheer number of computations that must be performed when tackling complex molecular systems. This problem is particularly severe in the realm of homogeneous catalysis, in which the reliable prediction of catalyst activity and selectivity often requires the optimization of hundreds of transition state (TS) structures.[1, 2, 3] Studies of potential new catalysts are often limited to only a few examples of a given reaction because finding so many TS structures for just one system requires a lot of time, and expanding the search to more variations of a given reaction is not always feasible within time constraints.

To address this and other challenges, we are developing a set of free, open source tools for structure manipulation and automation, collected under the organization QChASM (Quantum Chemistry Automation and Structure Manipulation, see Figure 1). QChASM currently comprises three main packages: AaronTools, AARON, and SEQCROW. AaronTools is a collection of tools (available as Perl modules or as a Python package) for building, measuring, manipulating, and comparing molecular structures; constructing input and parsing output files; submitting and monitoring jobs in high-performance computing environments; and analyzing data. AARON is a computational toolkit, written using AaronTools, to automate the geometry optimization of the many TS structures and energy minima required to predict the activity and selectivity of homogeneous catalytic reactions. Finally, SEQCROW is a plug-in for UCSF ChimeraX[4] that adds tools to build and modify complex molecular structures, map new catalysts and ligands onto previously-computed structures, manage AaronTools libraries, construct input files for quantum chemistry packages, and run and manage jobs.

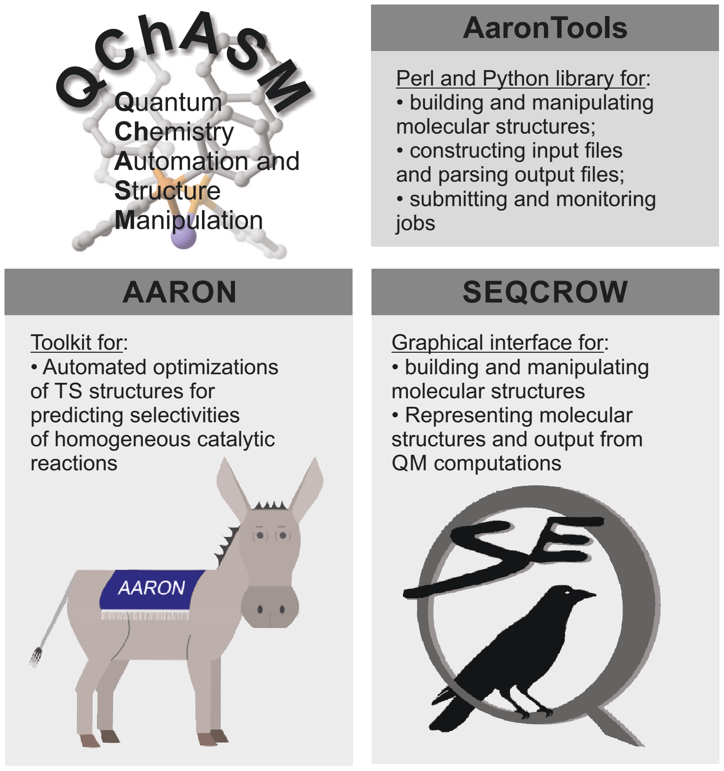

QChASM
Quantum Chemistry Automation and Structure Manipulation
AaronTools
Perl and Python library for:
• building and manipulating molecular structures;
• constructing input files and parsing output files;
• submitting and monitoring jobs
AARON
Toolkit for:
• Automated optimizations of TS structures for predicting selectivities of homogeneous catalytic reactions
AARON
SEQCROW
Graphical interface for:
• building and manipulating molecular structures
• Representing molecular structures and output from QM computations
SE

Figure 1: Main components of QChASM

These tools, described below, together facilitate the automation of quantum chemistry applications to complex molecular systems. Because AARON has been described recently[5] and a separate publication on SEQCROW is in preparation, we focus primarily on AaronTools, particularly the recent Python implementation. Our goal is to show that with these tools, users will be able to tackle more challenging problems by spending their time thinking about molecules instead of frittering away the hours building molecular structures, generating input files, submitting jobs, and parsing output files.

AARONTOOLS

In many quantum chemistry applications, a model system is used as a common framework to construct coordinates of many new molecules. For instance, you might consider a series of substituted analogs of a given molecule or need to optimize structures for different substrates or catalysts along the same reaction pathway. Typically, coordinates for such structures are generated using any of a number of available graphical molecular builders; however, this requires the user to make changes file by file, which is time consuming and can lead to mistakes, inconsistencies in atom numbering, etc. To avoid these mistakes, we have made central to AaronTools utilities to streamline the generation of complex molecular structures from either a simple script or the command line. These tools enable the generation of structures quickly and methodically, and are concise and flexible enough to be incorporated in workflows in many types of quantum chemistry applications, not limited to catalysis. There are two primary ways to use AaronTools, through either a series of stand-alone command-line scripts or the underlying Perl or Python objects.

Command Line Scripts

The AaronTools command line scripts provide basic functionality for structure measurement, manipulation, and comparison, as well as tools for output parsing and analysis. Most of these scripts provide a single specific function, such as modifying a distance, angle, or dihedral angle, adding or changing a substituent, or extracting thermochemical information from an output file. Internally, all structure manipulations are performed in Cartesian coordinates. Each command provides basic usage documentation and command line conveniences (e.g. file name globbing, pipes, and output redirection) are supported. In many cases, a simple shell script can be used to string together individual command line scripts to build up an automated workflow. A few examples are provided here (additional examples and usage help can be found on the AaronTools GitHub Wiki; see Additional Reading, below). While the Perl and Python AaronTools provide command line scripts with similar functionality, below we demonstrate the Python-based scripts.

A simple example using an AaronTools command line script is measuring a particular dihedral angle from a set of structures using the `dihedral` command with the `--measure` argument. For instance, the following will determine the dihedral angle defined by atoms 1, 2, 3, and 4 for each XYZ file in the current directory and print the value of the corresponding angle:

```
dihedral.py --measure 1 2 3 4 *.xyz
```

This and other command line scripts can also take output files from Gaussian,[6] Psi4,[7] or ORCA,[8] from which they will automatically extract the last set of molecular coordinates (e.g. from a geometry optimization).

A slightly more complex example uses `dihedral` to generate structures for a torsional scan. The `--set` argument accepts the four atoms defining the dihedral angle followed by the desired angle. Alternatively, the `--change` argument can be used to alter the selected torsional angle by the specified amount. We note that currently, `dihedral` can not change a dihedral angle involving four atoms within a ring and does not automatically resolve steric clashes or close contants that occur as the result of a change in dihedral angle. Combined with a simple `for` loop, the structures for each point along a torsional scan can be created and saved using the `--output` argument. For example, the following bash loop uses the coordinates from `original.xyz` to generate 35 new structures (each saved as a new XYZ file) by rotating the dihedral angle defined by atoms 1, 2, 3 and 4, in 10° increments:

```
for d in $(seq 0 10 350); do
   dihedral.py original.xyz --set 1 2 3 4 "$d" --output "rotated_$d.xyz"
done
```

Structures or components of structures can be translated using the `translate` command or rotated around any specified bond or axis using the `rotate` command. This includes simple rotations around the Cartesian axes as well as axes defined between centroids of groups of atoms or perpendicular to sets of atoms. For instance, Figure 2a shows the rotation of one component (atoms 20-34) of the stacked dimer in `dimer.xyz` by 60° around the axis perpendicular to the plane defined by atoms 20-30:

```
rotate.py dimer.xyz --targets 20-34 --perpendicular 20-30 --angle 60
```

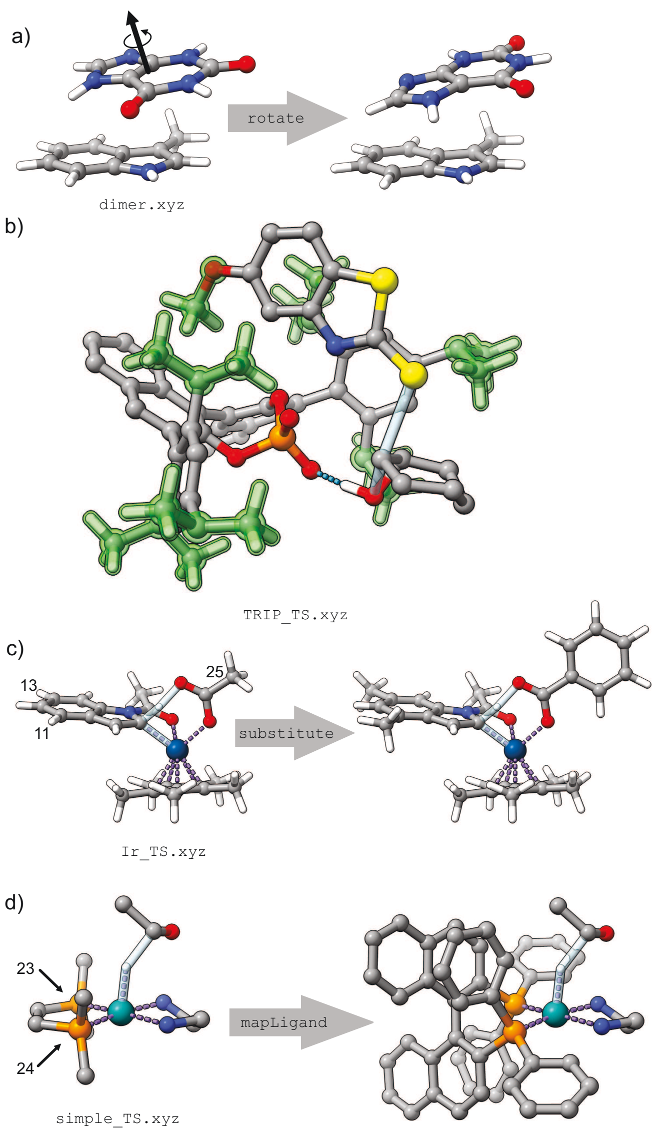


Figure 2: a) Dimer of 3-methylindole and 3,9-dihydro-purine-2,6-dione from Ref [9] before and after rotation of 3,9-dihydro-purine-2,6-dione by 60° around the vector normal to the ring plane; b) TS for a stereoselective epoxide ring opening catalyzed by the chiral phosphoric acid TRIP, from Ref [10], highlighting the six iPr and one OMe that will be rotated by `makeConf` (selected hydrogen atoms removed for clarity); c) TS for an Ir-catalyzed CH activation reaction before and after substitution of atoms 11 and 13 with Me groups and replacement of the Me group at atom 25 with a Ph ring using `substitute`; d) Use of `mapLigand` to replace a model ZDMP ligand with (S)-BINAP in a TS structure for a Noyori asymmetric hydrogenation of acetaldehyde.

A common task encountered in quantum chemistry applications is the generation of rotamers of substituents (e.g. iPr, OMe, etc). In systems with multiple rotatable groups, the number of potential conformations can be astronomical. The `makeConf` command automatically generates new conformers of a given structure by rotating all or selected substituents. AaronTools contains a library of common substituents along with information about the number of unique rotamers and the angles separating these rotamers. For example, in the AaronTools library a Ph substituent can exist as two rotamers, separated by 90°. Users can easily augment this library with their own, custom library, or override the number of rotamers considered for built-in substituents. `makeConf` uses this information to build rotameric structures. For instance, provided the TS structure shown in Figure 2b, `makeConf` will, by default, generate 2916 rotamers by considering three rotamers each of the six iPr groups and two rotamers of the OMe and OH groups. In this case, we would not need to consider rotations of the OH group, because we are confident that the OH—O hydrogen bond will be maintained in all low-lying TS structures. As such, we could use `makeConf` to automatically generate the 1458 rotamers by considering rotations of only the iPr and OMe groups, requesting that the generated conformers be written as separate XYZ files in a directory called `Conformers`:

```
makeConf.py TRIP_TS.xyz --substituent OMe iPr --output Conformers
```

Alternatively, specific iPr substituents could be rotated as identified by atom numbers. The structures generated by `makeConf.py` could then be optimized at a chosen level of theory to identify accessible TS structures, for example. Often, conformers generated by `makeConf` will exhibit steric clashes. Options allow for severe steric clashes to be automatically resolved or for such structures to simply not be printed.

The `substitute` command provides a flexible means of replacing any monovalent atom or substituent with a substituent from the built-in or user-defined libraries. To demonstrate this, Figure 2c shows the replacement of hydrogens 11 and 13 with methyl groups and the methyl group at atom 25 with a Ph ring in a TS structure for an Ir-catalyzed CH activation reaction via:

```
substitute.py Ir_TS.xyz -s 11,13=Me 25=Ph --minimize
```

With the `--minimize` argument, `substitute` will rotate the added substituents to minimize the Lennard-Jones energy, ensuring that new substituents are added in relatively low-energy orientations.

Additionally, `rmsdAlign` can be used to align two structures, minimizing the root mean squared deviation (RMSD) between the atom positions. This includes an efficient canonicalization scheme (triggered by the option `--sort`) that will automatically sort atoms within each structure to account for changes in numbering or degenerate rotations of substituents. For example, the following will calculate the RMSD between all XYZ files in a directory and a reference structure (`ref.xyz`), printing the results as comma-separated values:

```
rmsdAlign.py align/*.xyz --reference ref.xyz --sort -csv --output results.csv
```

AaronTools contains a library of nearly 100 achiral and chiral ligands as well as common organocatalysts. As with the substituent library, users can easily add their own custom ligand library. The `mapLigand` command provides an efficient means of replacing specified ligand(s) or catalyst(s) with other ones from either the built-in or individual user library. One could use `mapLigand` to take previously computed structures along a reaction pathway and replace the catalyst with another across each structure. The resulting geometries can then be optimized at the chosen level of theory to quickly explore reaction mechanisms for variations of a given reaction. As an example, Figure 2d shows the use of `mapLigand` to replace a simple bidentate ligand (ZDMP, bound to the metal through atoms 23 and 24) with the chiral ligand (S)-BINAP in a TS structure for a Noyori asymmetric hydrogenation of acetaldehyde:[11]

```
mapLigand.py simple_TS.xyz --ligand 23,24=S-BINAP
```

AaronTools can also extract thermochemical information from Gaussian (G09 or G16), ORCA, or Psi4 output files while also computing Grimme's quasi-RRHO[12] free energies. Options enable the user to automatically combine higher-level single point energies with thermochemical corrections computed at lower levels of theory, recalculate thermochemical corrections at a different temperature, and recursively search directories for output files. For example, the following extracts the energy from a set of single point jobs and combines these with enthalpy, RRHO free energy, and quasi-RRHO free energy corrections based on data extracted from the corresponding frequency jobs. The resulting data is printed to a `csv` file, along with warnings for any cases in which the single point and frequency jobs appear to use different geometries.

```
grabThermo.py --recursive *freq.log -sp *sp.log -csv --output thermo.csv
```

Finally, the commands `makeInput.py` and `submitJob.py` allow for the creation of Gaussian, Psi4, and Orca input files from the command line and submission to common queueing systems, respectively. For the creation of input files, level of theory and options are specified on the command line. Command line options also allow for the specification of bond constraints, DFT integration grids, etc., allowing the user to build sophisticated input files for these three computational chemistry packages all from the command line. Another command, `fetchMolecule.py` can generate initial molecular coordinates from SMILES or IUPAC names. As with all AaronTools command line scripts, output from one script can be read as STDIN for another, which makes it possible to fetch coordinates and place these directly into an input file. For instance, the following will build an Orca input file for the geometry optimization the structure of 1,3,5-trinitrotoluene at the B3LYP/def2TZVP level of theory:

```
fetchMolecule.py --iupac "1,3,5-trinitrotoluene" | makeInput.py --method b3lyp
--basis def2tzvp --optimize --output-format orca
```

The resulting input file could subsequently be submitted using `submitJob.py`.

Command line scripts are also available to measure or change inter-atomic distances and bond angles, change the element of a given atom (e.g. convert a C to an N, including automatic adjustment of hydrogen atoms to satisfy valency), change the chirality of a chiral center, calculate Sterimol parameters,[13] and follow imaginary vibrational modes, among others. These scripts provide command line access to common tasks encountered in quantum chemistry applications, allowing users to streamline and automate large portions of their workflows.

Scripting with Perl and Python

While the combination of AaronTools command line scripts enable the construction of automated workflows, more complex tools can be developed by directly using the objects implemented in the AaronTools modules (e.g. AARON and SEQCROW; see below). Some features of the Perl-based AaronTools have been described previously;[5] here, we provide a complementary description of the new Python AaronTools package.

The `Atom` class

Unlike the Perl version of AaronTools,[5] which uses indexed arrays within a `Geometry` object to keep track of atom information, the Python package introduces a new `Atom` class. Each `Atom` object contains all information associated with an individual atom, including the element and its coordinates as well as connections to other atoms, associated Lennard-Jones parameters, and tags useful for filtering and sorting. This data structure allows for more graceful and powerful interactions with molecular structures. For example, a molecular graph can be easily obtained by following the `Atom.connections` attributes throughout the structure. Additionally, the `Atom.tags` attribute allows AaronTools processes, as well as the programmer, to quickly identify structural components. Finally, extensibility is facilitated since sub-classes can be built on top of the `Atom` class without needing to redefine large swaths of code to use them. For instance, the `Atom` class was recently extended for use in parsing output files and writing input files for ONIOM computations,[14] which requires additional information to properly delineate regions of molecules that will be treated at different levels of theory.

The `Geometry` class

The most general way of interacting with a molecular structure is by using a `Geometry` object. Atoms and their coordinates are read from a number of common file formats, including XYZ files, Gaussian input and output files, and output files from Psi4[7] and ORCA[8] or from the AaronTools libraries. Atom connectivity is automatically determined. Utilities are available to transform or analyze the molecular structure or to prepare it for a workflow. For instance, one can perform substitutions, locate substituents by either name (e.g. Me, Ph, and iPr) or the atoms to which they are connected, determine the shortest path between specified atoms, identify molecular fragments, etc. After the desired changes are made, the `Geometry` object can be written to either an XYZ file or a Gaussian, ORCA, or Psi4 input file (see Managing Jobs, below).

The following illustrates the use of a `Geometry` object to add substituents by reading the coordinates of benzene and replacing atom 7 with a methyl group and atoms 8, 9, and 12 with NO_2 groups to generate TNT:

```
geom = Geometry('benzene.xyz')
geom.substitute('Me', '7')
o_and_p_positions = geom.find('8,9,12')
for position in o_and_p_positions:
   geom.substitute('NO2', position)
geom.write(outfile='tnt.xyz')
```

The above example used `find` to locate atoms by atom number, which obviously requires the programmer to know the atom numbering in the file `benzene.xyz`. However, `find` offers far more flexibility, allowing the programmer to locate atoms based on combinations of attributes including the element, the number and identity of bonded neighbors, and the distance from a given atom either spatially or connectively. As an example, the following will perform the same transformation of benzene to TNT, but without any prior knowledge of the atom ordering by first finding any hydrogen atom (h1), then locating the hydrogens that are three and five bonds away (the *o*- and *p*-hydrogens). The former is then substituted with Me and the latter with nitro groups.

```
geom = Geometry('benzene.xyz')
h1 = geom.find('H')[0]
o_and_p_positions = geom.find([BondsFrom(h1, 3), BondsFrom(h1, 5)], 'H')
geom.substitute('Me', h1)
for position in o_and_p_positions:
   geom.substitute('NO2', position)
geom.write(outfile='tnt.xyz')
```

Additionally, substituents can be detected and identified by name. For example, the following reads the geometry of TNT (`tnt.xyz`), detects all attached substituents, and then removes the methyl group using `remove_fragment` to leave 1,3,5-trinitrobenzene:

```
geom = Geometry('tnt.xyz')
geom.detect_substituents()
for sub in geom.substituents:
   if sub.name == 'Me':
      methyl_carbon = geom.find('Me', 'C')
      geom.remove_fragment(methyl_carbon, sub.end)
geom.write('trinitrobenzene.xyz')
```

In this example, `remove_fragment` requires the first and last atom of a fragment, so we use `sub.end` to request the last atom within the substituent.

Another feature of `Geometry` is the ability to add fused rings to structures. For example, one can convert benzene to naphthalene by first locating any hydrogen, then finding a second hydrogen three bonds away, and finally using `ring_substitute` to replace these two hydrogens with a new benzene ring:

```
geom = Geometry('benzene.xyz')
h1 = geom.find('H')[0]
h2 = geom.find(BondsFrom(h1, 3), 'H')[0]
geom.ring_substitute([h1, h2], 'benzene')
geom.write(outfile='naphthalene.xyz')
```

More complicated transformations can be achieved using combinations of `find`, `substitute`, and `ring_substitute`, all without knowledge of atom ordering.

Finally, the `Geometry` class provides the function `map_ligand` that allows the programmer to replace any specified ligand(s) with another ligand with the same total denticity. For instance, one can replace a bidentate ligand with another bidentate ligand, or a combination of a bidentate and monodentate ligand with a tridentate ligand. As an example, the following reads the coordinates of a TS structure for the enantioselective Markovnikov hydroboration of an alkene from `TS.xyz`,[15] locates the two phosphorus atoms of the original ligand using `find`, and then replaces this ligand with (S,S)-Me-DuPhos using `map_ligand` (see Figure 3):

```
oldcat = Geometry('TS.xyz')
Ps = oldcat.find('P')
oldcat.map_ligand('SS-Me-DuPhos', Ps)
```

In more complex cases (e.g. systems with additional P atoms or multiple ligands), one could locate the correct ligand atoms through the use of more refined attributes, as discussed above.

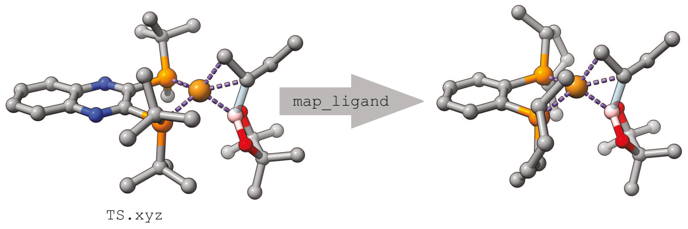


Figure 3: Use of the function `map_ligand` to replace the ligand in a TS for an enantioselective Markovnikov hydroboration (from Ref. [16]) of a terminal alkene with (S,S)-Me-DuPhos

The `Geometry` class provides many other functions, including computing Lennard-Jones energies, aligning structures for evaluating RMSDs, measuring and changing bond distances, angles, and dihedral angles, comparing connectivities between structures, canonicalizing atom ordering, identifying the shortest path between atoms, and performing substitutions, rotations, translations, etc. Information on these is readily available on the command line through `pydoc`.

Managing Jobs

Finally, AaronTools provides functions to create input files for Gaussian, ORCA, and Psi4 and to submit and monitor jobs using popular queuing software (Slurm, Torque/MOAB, LSF, and SGE). Combined with the structure manipulation capabilities described above, these functions enable the development of complex toolkits and automated workflows. For this purpose, a `Theory` class provides an intuitive means of setting the level of theory as well as job type (optimization, vibrational frequencies, etc), charge and multiplicity, and other job-specific options (number of processors, integration grids for DFT computations, etc). A `Theory` object can contain a `BasisSet` object to facilitate specification of more complex basis sets (e.g. ECPs, auxiliary basis sets, etc.). Once constructed, a `Theory` object can be passed to a `Geometry` object in order to automatically construct the corresponding input file for the chosen software package. The resulting input file can then be submitted using a `SubmitProcess` object, with an option to await all running jobs in the current directory to finish before proceeding.

As an example, the following will read the geometry from `dimer.xyz` (e.g. Figure 2a) and optimize this structure using Gaussian at the ωB97X-D/def2-TZVP level of theory. Once this job is complete, the script reads the optimized geometry and then uses ORCA to run a DLPNO-CCSD(T)/cc-pVQZ single point.

```
#read initial coordinates
geom = Geometry("dimer.xyz")
#method for optimization
dft_theory = Theory(method="wB97X-D", basis="def2-TZVP", processors=14, memory=24,
job_type=OptimizationJob())
#write optimization input file and submit, waiting for job to finish
geom.write(outfile="opt.com", theory=dft_theory)
opt_job = SubmitProcess(fname="opt.com", walltime=24, processors=14, memory=28)
opt_job.submit(wait=True)

#read the optimized structure
opt_geom = Geometry("opt.log")
#basis set and method for DLPNO-CCSD(T) single point
dlpno_basis = BasisSet([Basis("cc-pVQZ"), Basis("cc-pVQZ", aux_type="C")])
dlpno_theory = Theory(method="DLPNO-CCSD(T)", basis=dlpno_basis, processors=4,
memory=28, job_type=SinglePointJob())
#Write DLPNO-CCSD(T) input file and submit
opt_geom.write(outfile="dlpno.inp", theory=dlpno_theory, simple=["TightSCF"])
dlpno_job = SubmitProcess(fname="dlpno.inp", walltime=24, processors=4, memory=32)
dlpno_job.submit(wait=True)
```

SIDEBAR: Remote Job Submission with FireWorks

We are currently extending AaronTools to work with the FireWorks workflow software,[17] which will allow more rapid deployment of AaronTools-based tools across different HPC environments. FireWorks is a free, open-source toolkit for managing HPC workflows over arbitrary computing resources, including those that have queueing systems. As such, the integration of AaronTools and AARON with FireWorks will also enable remote job submission across multiple computer clusters and job monitoring and management via a web interface. This added functionality will be part of the new Python-based AARON (AARON2), which is currently in development.

AARON

One example of a complex tool developed using the Perl-based AaronTools is AARON, which has been described in detail before.[5] Briefly, AARON interfaces with Gaussian09 or Gaussian16 to automate the optimization of TS structures derived from templates through substitutions or changes of ligand/organocatalyst. First, the user provides the TS configurations to be considered in the form of a TS library (i.e. template structures leading to the *R*- or *S*-product of an enantioselective reaction), or they are requested from the built-in template library. Next, any requested changes to the template are handled, such as mapping a new ligand or making substitutions. Additionally, conformers of rotatable groups are sampled automatically. After structure generation, an initial refinement is performed, freezing the unchanged parts of the structure and relaxing the new additions, generally at an inexpensive level of theory (e.g. semi-empirical). Next, geometry optimizations at the density functional theory (DFT) level are done, first with any forming or breaking bonds frozen, and then with no constraints. After a full geometry optimization, harmonic frequencies and thermochemistry can be computed. If structures exhibit the wrong number of imaginary vibrational modes, AARON will attempt to automatically resolve the problem through additional constrained and unconstrained optimizations. If desired, a final single-point computation can be done at a higher level of theory, which is then combined with the thermochemical corrections derived from the original level of theory. While all these computations are running, AARON is continuously monitoring their status — handling common errors, resubmitting failed jobs, and resolving unexpected structural changes (for example, by constraining a problematic bond) — all without user intervention. Finally, after all computations are finished, AARON parses the output files, analyzes the thermochemistry to generate a summary of the results and predict product ratios based on a Boltzmann weighting of accessible pathways, and generates the associated Supporting Information.

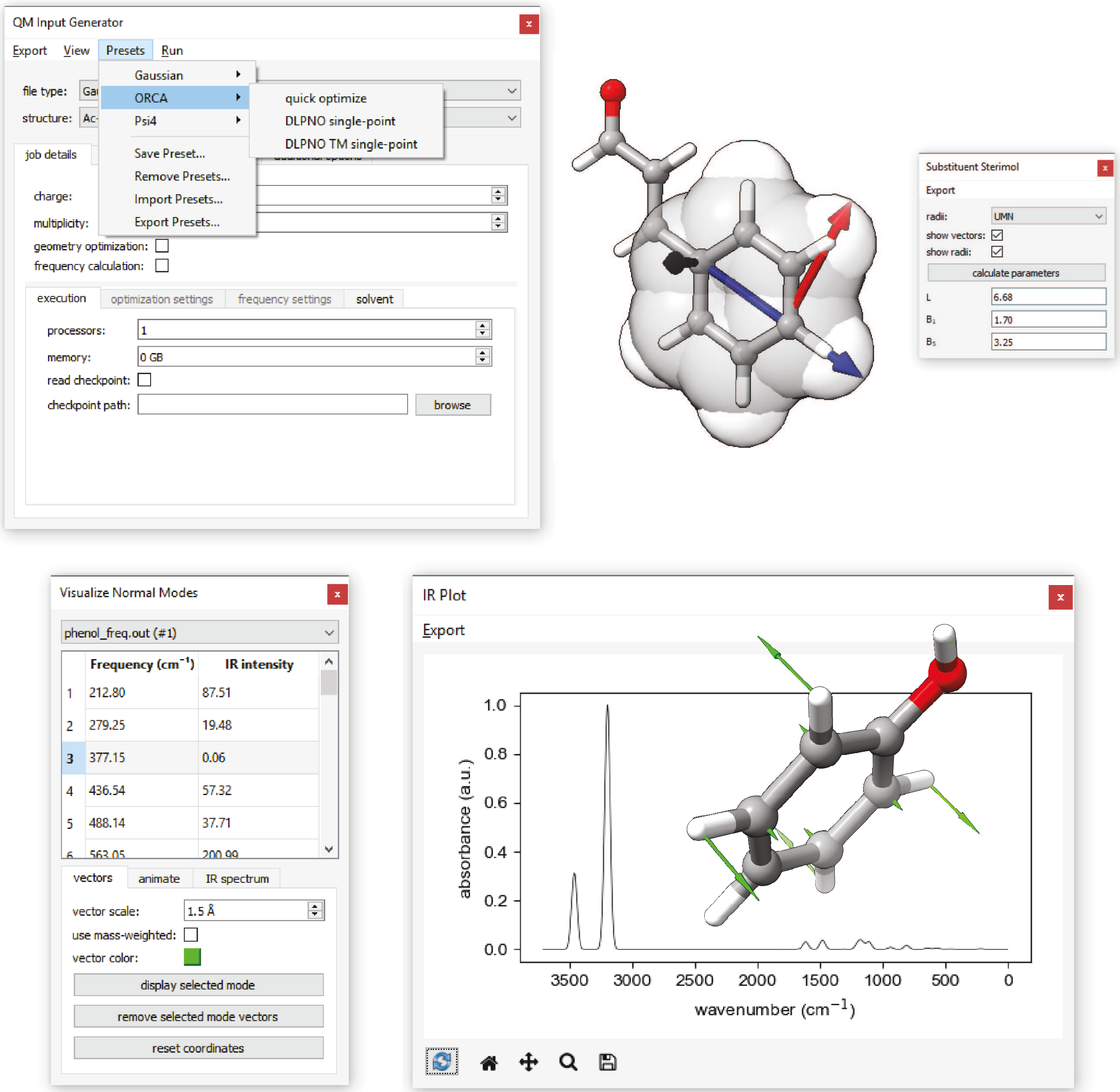
The primary use of AARON is to accelerate predictions of the activity and selectivity of homogeneous catalysts by automating optimization of TS structures. This automation allows computational predictions of selectivities for more examples of a given reaction than could reasonably be done without it. This opens the door for a number of exciting possibilities, including virtually screening catalysts with DFT methods and benchmarking DFT methods on statistically significant numbers of catalysts. AARON has been applied to a number of catalytic reactions,[18, 19, 20] several of which were recently summarized[5] and is currently being used to benchmark DFT methods for an Ir-catalyzed CH activation reaction (See Figure 2c). Overall, the automation provided by AARON brings us closer to true computationally-driven rational catalyst design. The current version of AARON[5] is built using the Perl-based AaronTools. A more powerful Python version of AARON (AARON2) is currently in development that will exploit the added flexibility of the Python implementation of AaronTools.

SIDEBAR: Automated Conformer Searches using XTB/Crest

Vital to successful applications of AARON is the generation of all reasonable configurations and conformers in the construction of a TS template library. We have had recent success doing this by combining Grimme's XTB/Crest automated conformational search[21] with AARON. For instance, starting from one conformation of each configuration, Crest can be used to generate a collection of conformations for an initial TS template library (constraining any forming or breaking bonds), which can then be further refined using AARON. This avoids the often cumbersome task of enumerating conformations when constructing a TS template library, particularly in cases of ring-flips and other more complex conformational changes that are not handled automatically by AARON.

SEQCROW

While the main utility of AaronTools is the ability to build and manipulate complex molecular structures from the command line or from within a Perl or Python script, use of a graphical interface is often advantageous. While graphical molecular builders abound, few are well suited for the rapid construction of the structures encountered in studies of homogeneous catalysis. We are developing a plug-in for UCSF ChimeraX[4] called SEQCROW that provides the power of AaronTools within a graphical user interface. For example, SEQCROW can be used to generate and explore potential TS structures for transition metal catalyzed reactions using different chiral ligands or perform substitutions and add fused rings to structures. SEQCROW also contains graphics presets for the generation of publication-quality images of small to medium sized molecular structures and extends the capabilities of ChimeraX to be able to generate input files for popular quantum chemistry packages and run these jobs, plot simulated spectra, calculate thermal energy corrections, etc. (see Figure 4). SEQCROW is still in development, but is available through the ChimeraX 'Toolshed.' It will be described more fully in a forthcoming publication.



file type: Ga
structure: Ac
charge:
multiplicity:
geometry optimization: ☐
frequency calculation: ☐
processors: 1
memory: 0 GB
read checkpoint: ☐
checkpoint path:

radii: UMN
show vectors: ☑
show radii: ☑

L	6.68
B_1	1.70
B_5	3.25

phenol_freq.out (#1)

	Frequency (cm^{-1})	IR intensity
1	212.80	87.51
2	279.25	19.48
3	377.15	0.06
4	436.54	57.32
5	488.14	37.71
6	563.05	200.99

vector scale: 1.5 Å
use mass-weighted: ☐
vector color: 🟩

Figure 4: Screenshots of SEQCROW showing the QM Input Generator allowing users to save 'Presets' of common input file types for Gaussian, Orca, and Psi4, the calculation of Sterimol parameters, and the Normal Mode Visualization and IR Spectra Plotting

CONCLUSIONS

As quantum chemists have gravitated to larger and more complex molecular systems, new challenges have emerged regarding the execution of large numbers of computations, parsing the corresponding output files, and verifying and managing the resulting data. In the context of catalysis, reliable predictions of catalyst performance often require consideration of hundreds of potential TS structures. The time required to generate input files, run the required jobs, and check the resulting output files often prevents the assessment of large numbers of potential catalysts, precluding effective computational catalyst design. We have developed a suite of open-source tools (QChASM) to meet some of these challenges. The workhorse of QChASM is AaronTools, which consists of Perl and Python tools for building, measuring, and manipulating molecular structures. Using AaronTools, one can quickly and intuitively perform substitutions or exchange chiral and achiral ligands, submit and monitor jobs, and construct input and parse output files from popular electronic structure packages. AARON is a more specialized toolkit for automated predictions of selectivities for catalytic reactions. Finally, SEQCROW provides a graphical interface to exploit the power of AaronTools, build input files for QM computations, and generate molecular images.

While these tools were initially developed for applications in computational catalysis, they should be of general use by the quantum chemistry community, AaronTools being particularly broad in its applicability. For example, we have relied extensively on AaronTools in our studies of stacking interactions and supramolecular complexes.[22] Furthermore, as free, open-source tools, contributions from others are welcomed in order to build even more powerful and useful tools for the broader computational chemistry community.

ACKNOWLEDGEMENTS

We thank B. J. Rooks and Y. Guan for their efforts in the development of AARON and the Perl-based AaronTools as well as M. R. Haas, A. C. Doney, and A. N. Bootsma for feedback and testing. Molecular graphics were generated in UCSF ChimeraX, developed by the Resource for Biocomputing, Visualization, and Informatics at the University of California, San Francisco. We have no conflicts of interest to declare.

FUNDING INFORMATION

This work was funded by National Science Foundation Grants CHE-1266022 and CHE-1665407.

References

[1] Santoro S, Kalek M, Huang G, Himo F. Elucidation of Mechanisms and Selectivities of Metal-Catalyzed Reactions using Quantum Chemical Methodology [Journal Article]. Acc Chem Res. 2016;49(5):1006–18. Available from: https://www.ncbi.nlm.nih.gov/pubmed/27082700.

[2] Vitek AK, Jugovic TME, Zimmerman PM. Revealing the Strong Relationships between Ligand Conformers and Activation Barriers: A Case Study of Bisphosphine Reductive Elimination [Journal Article]. ACS Catalysis. 2020;10(13):7136–7145.

[3] Foscato M, Jensen VR. Automated in Silico Design of Homogeneous Catalysts [Journal Article]. ACS Catalysis. 2020;10(3):2354–2377.

[4] Goddard TD, Huang CC, Meng EC, Pettersen EF, Couch GS, Morris JH, et al. UCSF ChimeraX: Meeting modern challenges in visualization and analysis [Journal Article]. Protein Sci. 2018;27(1):14–25. Available from: https://www.ncbi.nlm.nih.gov/pubmed/28710774.

[5] Guan Y, Ingman VM, Rooks BJ, Wheeler SE. AARON: An Automated Reaction Optimizer for New Catalysts [Journal Article]. J Chem Theory Comput. 2018;14(10):5249–5261. Available from: https://www.ncbi.nlm.nih.gov/pubmed/30095903.

[6] Frisch MJ, Trucks GW, Schlegel HB, Scuseria GE, Robb MA, Cheeseman JR, et al.. Gaussian˜16 Revision C.01; 2016. Gaussian Inc. Wallingford CT.

[7] Turney JM, Simmonett AC, Parrish RM, Hohenstein EG, Evangelista FA, Fermann JT, et al. PSI4: an open-source ab initio electronic structure program [Journal Article].

Wiley Interdisciplinary Reviews-Computational Molecular Science. 2012;2(4):556–565. Available from: <GotoISI>://WOS:000305393700004.

[8] Neese F. Software update: the ORCA program system, version 4.0 [Journal Article]. Wiley Interdisciplinary Reviews-Computational Molecular Science. 2018;8(1). Available from: <GotoISI>://WOS:000418158400001.

[9] Bootsma AN, Doney AC, Wheeler SE. Predicting the Strength of Stacking Interactions between Heterocycles and Aromatic Amino Acid Side Chains [Journal Article]. Journal of the American Chemical Society. 2019;141(28):11027–11035. Available from: https://doi.org/10.1021/jacs.9b00936.

[10] Seguin TJ, Wheeler SE. Electrostatic Basis for Enantioselective Bronsted-Acid-Catalyzed Asymmetric Ring Openings of meso-Epoxides [Journal Article]. ACS Catalysis. 2016;6(4):2681–2688. Available from: https://doi.org/10.1021/acscatal.6b00538.

[11] Sandoval CA, Ohkuma T, Muniz K, Noyori R. Mechanism of asymmetric hydrogenation of ketones catalyzed by BINAP/1,2-diamine-ruthenium(II) complexes [Journal Article]. J Am Chem Soc. 2003;125(44):13490–503. Available from: https://www.ncbi.nlm.nih.gov/pubmed/14583046.

[12] Grimme S. Supramolecular binding thermodynamics by dispersion-corrected density functional theory [Journal Article]. Chemistry A European Journal. 2012;18(32):9955–64. Available from: https://www.ncbi.nlm.nih.gov/pubmed/22782805.

[13] Verloop A, Hoogenstraaten W, Tipker J. Chapter 4 - Development and Application of New Steric Substituent Parameters in Drug Design. Drug Design. 1976:165 – 207.

[14] Vreven T, Byun KS, Komáromi I, Dapprich S, Montgomery JA, Morokuma K, et al. Combining Quantum Mechanics Methods with Molecular Mechanics Methods in ONIOM [Journal Article]. Journal of Chemical Theory and Computation. 2006;2(3):815–826. Available from: https://doi.org/10.1021/ct050289g.

[15] Iwamoto H, Imamoto T, Ito H. Computational design of high-performance ligand for enantioselective Markovnikov hydroboration of aliphatic terminal alkenes [Journal Article]. Nature Communications. 2018;9(1):2290. Available from: https://doi.org/10.1038/s41467-018-04693-9.

[16] Iwamoto H, Imamoto T, Ito H. Computational design of high-performance ligand for enantioselective Markovnikov hydroboration of aliphatic terminal alkenes [Journal Article]. Nat Commun. 2018;9(1):2290. Available from: https://www.ncbi.nlm.nih.gov/pubmed/29895938.

[17] Jain A, Ong SP, Chen W, Medasani B, Qu X, Kocher M, et al. FireWorks: a dynamic workflow system designed for high-throughput applications. Concurrency and Computation: Practice and Experience. 2015;27(17):5037–5059. CPE-14-0307.R2. Available from: http://dx.doi.org/10.1002/cpe.3505.

[18] Wheeler SE, Seguin TJ, Guan Y, Doney AC. Noncovalent Interactions in Organocatalysis and the Prospect of Computational Catalyst Design [Journal Article]. Acc Chem Res. 2016;49(5):1061–9. Available from: https://www.ncbi.nlm.nih.gov/pubmed/27110641.

[19] Guan Y, Wheeler SE. Automated Quantum Mechanical Predictions of Enantioselectivity in a Rhodium-Catalyzed Asymmetric Hydrogenation [Journal Article]. Angew Chem Int Ed Engl. 2017;56(31):9101–9105. Available from: https://www.ncbi.nlm.nih.gov/pubmed/28586140.

[20] Doney AC, Rooks BJ, Lu T, Wheeler SE. Design of Organocatalysts for Asymmetric Propargylations through Computational Screening [Journal Article]. ACS Catalysis. 2016;6(11):7948–7955.

[21] Pracht P, Bohle F, Grimme S. Automated exploration of the low-energy chemical space with fast quantum chemical methods. Phys Chem Chem Phys. 2020;22:7169–7192. Available from: http://dx.doi.org/10.1039/C9CP06869D.

[22] Wheeler SE. Understanding Substituent Effects in Noncovalent Interactions Involving Aromatic Rings [Journal Article]. Accounts of Chemical Research. 2013;46(4):1029–1038. Available from: https://doi.org/10.1021/ar300109n.

FURTHER READING

QChASM GitHub

https://github.com/QChASM

AaronTools GitHub Wiki (Perl)

https://github.com/QChASM/AaronTools/wiki

AaronTools GitHub Wiki (Python)

https://github.com/QChASM/AaronTools.py/wiki