SMARTpy: A Python Package for the Generation of Cavity Steric Molecular Descriptors and Applications to Diverse Systems

Beck R. Miller, Ryan C. Cammarota, Matthew S. Sigman*

Department of Chemistry, University of Utah, 315 South 1400 East, Salt Lake City, UT 84112, USA

Table of Contents

1. Supplemental Figures and Tables

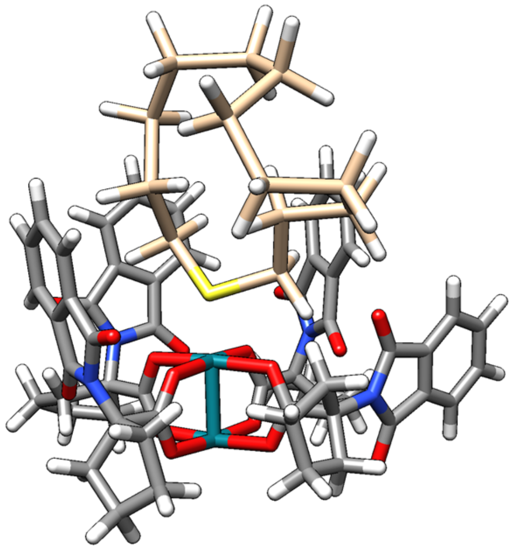

Figure S1. Example outlier for V_{CAVITY}. SMART cavity ensembles only have a single conformer.

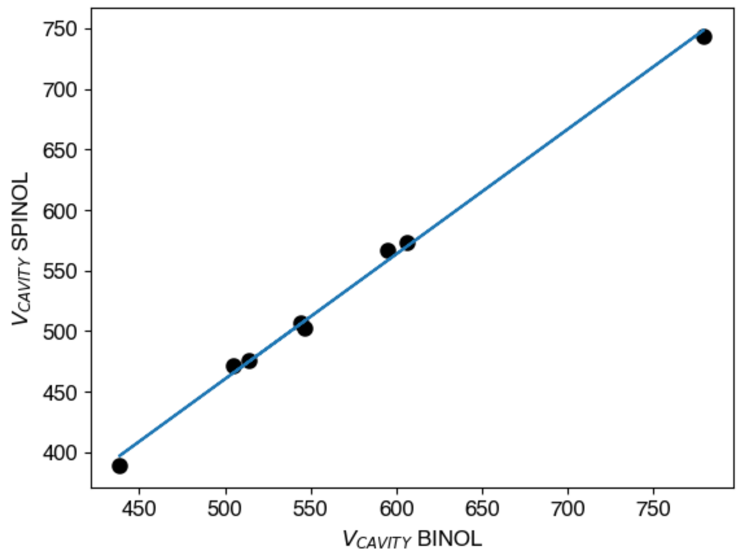

Figure S2. V_{CAVITY} for BINOL and SPINOL are linearly correlated.

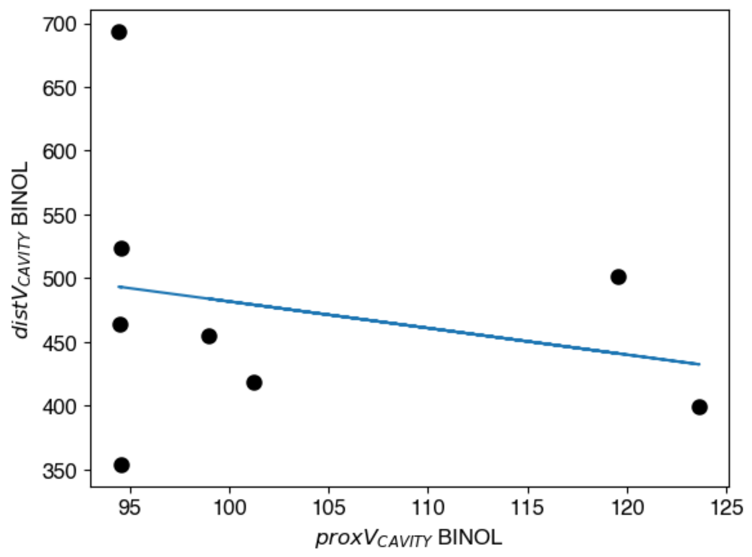

Figure S3. No correlation between distV$_{\text{CAVITY}}$ and proxV$_{\text{CAVITY}}$ for BINOL scaffolds.

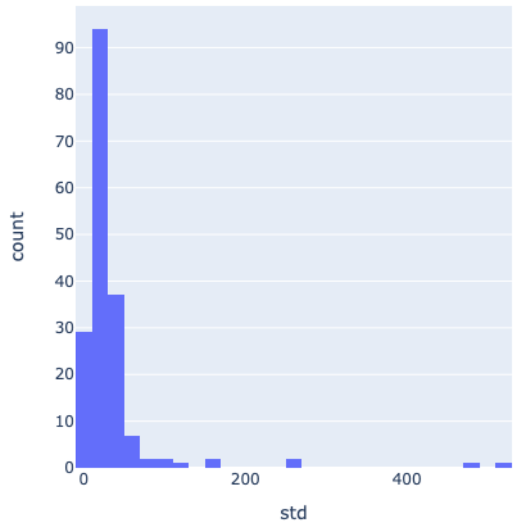

Figure S4. Standard deviation (std) between three runs of SMARTpy using NSTEP=50.

Table S1. Times for Computing SMART Descriptors via Two Different Methods.

Method	Backend	Total Time (s)*	Average Time (s)**
1	PyVista	4.04	0.22
2	Morfeus	215	12.0

*Total time to compute descriptors for 18 structures.

**Average time taken per structure.

Table S2. Probe Structures Used for Analysis

Name	**Filename***
CycH_8	S_SiF2_8_cyclic.mol2
CycH_10	S_SiF2_10_cyclic.mol2
CycH_12	S_SiF2_12_cyclic.mol2
AcycH_8	S_SiF2_8_acyclic.mol2
AcycH_10	S_SiF2_10_acyclic.mol2
AcycH_12	S_SiF2_12_acyclic.mol2
CycF_12	S_SiF2_12_cyclic.mol2
LinH_6	O_SiH2_6_linear.mol2

*Probe structures used in this manuscript and other general probe structures stored in https://github.com/SigmanGroup/SMART-molecular-descriptors/tree/main/Probes

Table S3. SMART Molecular Descriptors Supported by Python Workflow

Manuscript Abbrev.	**Feature Description**
$\mathbf{V_{CAVITY}}$	Cavity volume
$\mathbf{A_{CAVITY}}$	Cavity surface area
CSA	Contact surface area
ESA	Entry surface area (ESA=A_{CAVITY}-CSA)
prox$\mathbf{V_{CAVITY}}$	Proximal cavity volume (radius = i)
prox$\mathbf{A_{CAVITY}}$	Proximal cavity surface area (radius = i)
proxCSA	Proximal contact surface area
proxESA	Proximal entry surface area ($proxESA_{r=i} = proxA_{CAVITY,r=i} - proxCSA_{r=i}$)
L	Sterimol L (height)
B1	Sterimol B1 (min width)
B5	Sterimol B5 (max width)

2. Computational Details for the SMART Python Workflow

2.1. Aligning the Molecular Probe to a Structure

The SMARTpy workflow is initiated by the definition of a binding vector between the molecular probe and a computed structure of interest. An algorithm aligns two vectors, from the probe and structure and computes a binding site for the probe. The probe binding vector is defined by the tether atom and a dummy H atom that is removed after docking (Figure S5a) (See Section 4 for more information on defining probes). Multiple implementations for defining a structure binding vector are executed for maximal system modularity.

The first method is a linear tail-to-tip vector, defined by the binding atom of the structure and a reference atom located along the desired binding axis (Figure S5b). The distance (d) between the structure and probe binding site is specified by the user (default to 2.0 Å). This method reflects the original implementation on dirhodium(II) cores, where the linear core defined the axial binding vector. The second method is a cross-product computation method for binding perpendicular to an angle of reference given a central binding atom and two reference atoms (Figure S5c). This method was designed for planar complexes with an axial active site, such as Fe-porphyrins. The reference vector is the cross product of the angle defined between the two reference atoms with respect to the binding atom. The third method was developed specifically to accommodate metal coordination geometry that eludes the previous two reference methods, such as tetrahedral and trigonal planar. A user may define ligand atoms bound to the docking site and a desired geometry, and the position of the empty coordination site is computed with respect to the defined binding atoms (Figure S5d). The fourth method places the probe at a defined binding coordinate. This method was implemented for applications of replacing substructure atoms with a molecular probe (Figure S5e). This method requires the definition of one or more atoms to calculate a reference vector position.

Finally, a geometry auto-detection method was implemented for ease of user application. A user may define the ligand bond type (covalent or dative) that dictates the parameters of the subsequent search. For covalently bound complexes, the neighbors of the binding atom are detected using RDKit[1] (Figure S5f). For datively bound or mixed complexes, a search radius may be defined about the binding site (defaulting to the binding atom Van der Waals radius) for detection of proximity-based neighbors. The same geometry analysis defined previously is implemented to identify the position of the empty coordination site. In all methods, the probe coordinates are translated to the calculated binding position and rotated to align with the structure vector (Figure S5g). The docked probe is assessed for steric clashes with the structure using atomic Van der Waals radii. The binding atom of the structure is disregarded in the clash detection process.

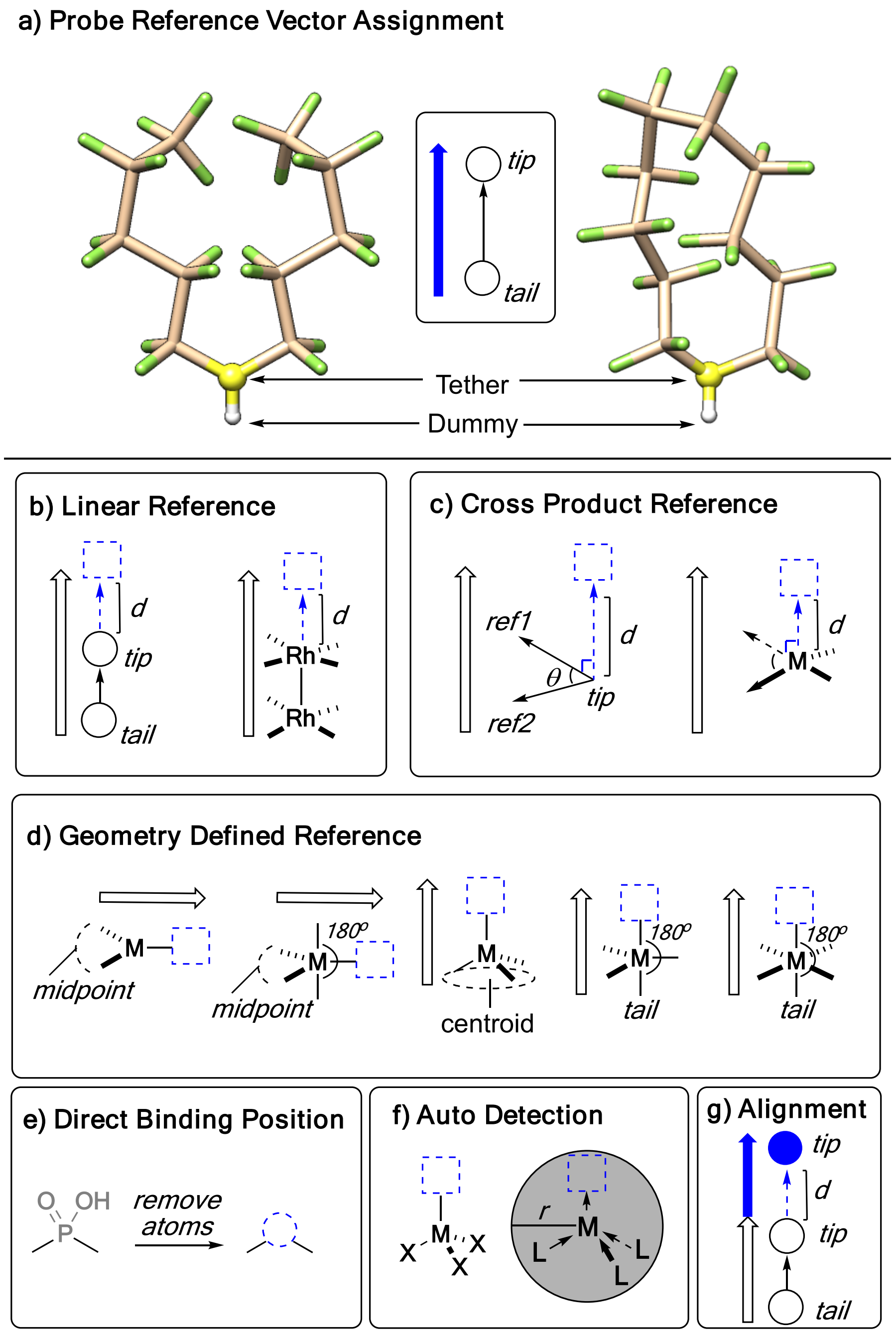

Figure S5: Reference vectors for: a) cyclic and acyclic probes, and (b-f) various structure reference methods. g) Alignment of two vectors for docking.

2.2. Probe Template Generation and Conformational Searching

Template-based conformer searching methods relied on the matching of specific substructures to previously generated templates.[20] Since the molecular probe is generally small, the whole molecular probe structure is considered during template generation. Probe conformers are freely generated using RDKit[1] to generate a conformational template. Each conformer is aligned to the binding axis and binding position on the original structure and assessed for atomic overlap (steric clashes) with the structure. Electrostatic interactions are not considered in this approach, allowing for a rapid search method for full cavity exploration. If no such clash is detected, a copy of the probe conformer is saved, and the original conformer is rotated about the docking axis by a randomly generated displacement. A user may define how many iterations this algorithm completes before returning the compiled probe ensemble.

2.3. Computation of Molecular Descriptors

SMART descriptors are computed from the compiled probe conformational ensemble. The original workflow uses UCSF Chimera[2] to compute these properties, but the SMARTpy can additionally apply three different methods for descriptor calculation, each with unique advantages. Comparison of the descriptors from each method will be analyzed below.

The first method uses the Python API for PyVista[3] to compute a surface enclosing the probe ensemble. Conformer atoms are regarded as XYZ coordinate vertices in space comprising a 3D "point cloud". Triangulation is used to identify and connect the vertices at the outer edges of the point cloud, generating a representative surface area. This method is directly analogous to the alpha shape method and allows for rapid computation of SMART descriptors for an irregular 3D shape. Proximal measurements are also computed via this method by intersecting the point cloud with a sphere of a defined radius and computing the intersection volume ($\text{proxV}_{\text{CAVITY}}$) and intersection surface area ($\text{proxA}_{\text{CAVITY}}$). Distal volume is defined as the difference between the full and proximal measurements.

One limitation to this method is that the radii of the atoms are not considered in triangulation. As a result, the absolute surface area and volume will be smaller compared to methods that consider atomic radii. Another limitation arises from the requirement for careful selection of the parameter alpha, which defines the search radius around each point for connection on the generated surface (See PyVista documentation for more information). This parameter determines the resolution of the pocket surface.

The second method utilizes the Python suite Morfeus[4] to obtain surface area from solvent accessible surface area (SASA) and compute volume directly from V_{Bur}. SASA is first computed for each component of the system, the full complex, the probe ensemble, and the structure (Figure SXa). The contact SASA is defined as the difference between the complex SASA and the sum of the component SASAs (Figure SXb). A maximum radius is computed between the structure binding point and the farthest probe conformer atom, generating a sphere encompassing the entire probe ensemble. V_{Bur} is then calculated to describe the absolute V_{CAVITY} occupied by the ensemble within the maximal sphere. Proximal descriptors can be calculated using this method by decreasing the radius of the V_{Bur} calculation. Maximal and minimal width of the probe ensemble is also defined using Sterimol descriptors. Quadrant and octant analysis can also be employed through Morfeus (https://digital-chemistry-laboratory.github.io/morfeus/buried_volume.html), using atoms identifying the z-axis and xz-plane as defined by Cavallo et. al.[5]

3. SMART API Usage Guide

***NOTE: This section is meant to serve as a general guide to the order of operation within the workflow. See the GitHub repository or readthedocs.org (https://smart-molecular-descriptors.readthedocs.io/en/latest/) for current version syntax and examples of usage.

Python dependencies:

- RDKit[1]
- numpy
- pandas
- scipy
- morfeus (optional)
- pyvista (optional)

3.1. Defining Binding Vectors

An input structure file must first be supplied by the user. This file is converted into a `STRUCTURE` object using either the function `ReadStructure()` or `ReadMol()`, for input files or RDKit MOL objects respectively. File types of SDF, MOL, MOL2, PDB, and XYZ are supported. Similarly, a molecular probe is initialized as a `PROBE` object using the function `Probe()`, specified by its direct filename (ex: `Probe("S_SiF2_8_acyclic.mol2")`). Probe files are required to be in MOL2 format and are stored in and can be read directly from the package directory `Probes/`. A path to other probe file locations can also be supplied.

3.2. Probe Cavity Ensemble Generation

Once all inputs and reference vectors are specified, the function `run()` is called to perform the docking. This function returns a `DOCK` class object that can be used directly in the next step. An optional utility for saving the docked structure as a MOL file is implemented in `ExportDocked()` and is available for debugging.

<u>Setting Conformational Search Parameters:</u>

- `NSTEP` defines the number of steps the algorithm takes to generate conformers.
- `MAXROTATION` and `MINROTATION` are used to define the maximum and minimum allowed rotations in degrees for the search algorithm respectively.
- `SEED` is available to supply an integer seed to the code for reproducibility.

The first step of the template search algorithm is to generate a conformer "template" for the desired probe. This is achieved through the RDKit command `EmbedMultipleConformers()` with the desired number of conformers set to 200. These templates can also be loaded from MOL2 files in instead of being autogenerated. The docked probe (if utilized) is removed from the structure and each conformer of the template ensemble is translated to the docked probe position, matched by the tether atom, and assessed for steric clashes with the structure. If no steric clash is detected, the conformation is saved into an ENS object and rotated about the reference axis by a randomly generated displacement. Else, only rotation occurs. This process repeats with the full template ensemble for `NSTEP`. This parameter is recommended to be set around 20 for template searching, which is the code default. It is recommended to visually inspect the ensemble generated in a test case, to ensure that `NSTEP` is set correctly.

If the pocket is not sufficiently explored or resultant pockets are too sparse, consider increasing `NSTEP`

- See pyvista documentation for a definition of `alpha` (default = 0). (https://docs.pyvista.org/version/stable/api/)
- See Morfeus documentation for a description of `xz_plane_atoms`. (https://digital-chemistry-laboratory.github.io/morfeus/buried_volume.html)

4. Instructions for Customization Probe Generation

The SMART workflow has been redesigned with broader applications in mind than the initial dirhodium(II) applications. As a result, users may wish to use more specific probes to match a target substrate or probes constructed with different atoms. The Python package is distributed with general Si-core probes in the Probes/ subdirectory. Any customized probes placed in this folder or identified through an absolute path may be called in the code.

To generate a structure that can be used as a probe technical detail must be considered. The probe structures must be in MOL2 format, and the first atom listed in the file is designated the "tether" atom. The last atom in the file should be a dummy H that defines the probe vector to align to a structure. *This atom will be removed before steric clash assessment and from final docked structures.*

5. References

1. RDKit: Open-source cheminformatics. https://www.rdkit.org

2. UCSF Chimera--a visualization system for exploratory research and analysis. Pettersen EF, Goddard TD, Huang CC, Couch GS, Greenblatt DM, Meng EC, Ferrin TE. *J Comput Chem.* 2004 Oct;25(13):1605-12.

3. Sullivan et al., (2019). PyVista: 3D plotting and mesh analysis through a streamlined interface for the Visualization Toolkit (VTK). Journal of Open Source Software, 4(37), 1450, https://doi.org/10.21105/joss.01450

4. https://digital-chemistry-laboratory.github.io/morfeus

5. Laura Falivene, Raffaele Credendino, Albert Poater, Andrea Petta, Luigi Serra, Romina Oliva, Vittorio Scarano, and Luigi Cavallo. SambVca 2. A web tool for analyzing catalytic pockets with topographic steric maps. *Organometallics*, 35(13):2286–2293, 2016. doi:10.1021/acs.organomet.6b00371