rpxdock Documentation

Release 0.0.1

Will Sheffler

Apr 30, 2020

TABLE OF CONTENTS

CHAPTER

ONE

INTRO

need some gentle introduction here…

CHAPTER

TWO

INSTALLATION

pip install git@github.com:willsheffler/rpxdock.git

or for dev,

git clone git@github.com:willsheffler/rpxdock.git my_rpxdock_code cd my_rpxdock_code conda env update pytest (will take a few min to build cpp files) (check that all tests pass…)

email willsheffler@gmail.com when nothing works

CHAPTER

THREE

TUTORIALS

3.1 A dummy tutorial

uh. . . . import rpxdock I guess?

CHAPTER FOUR

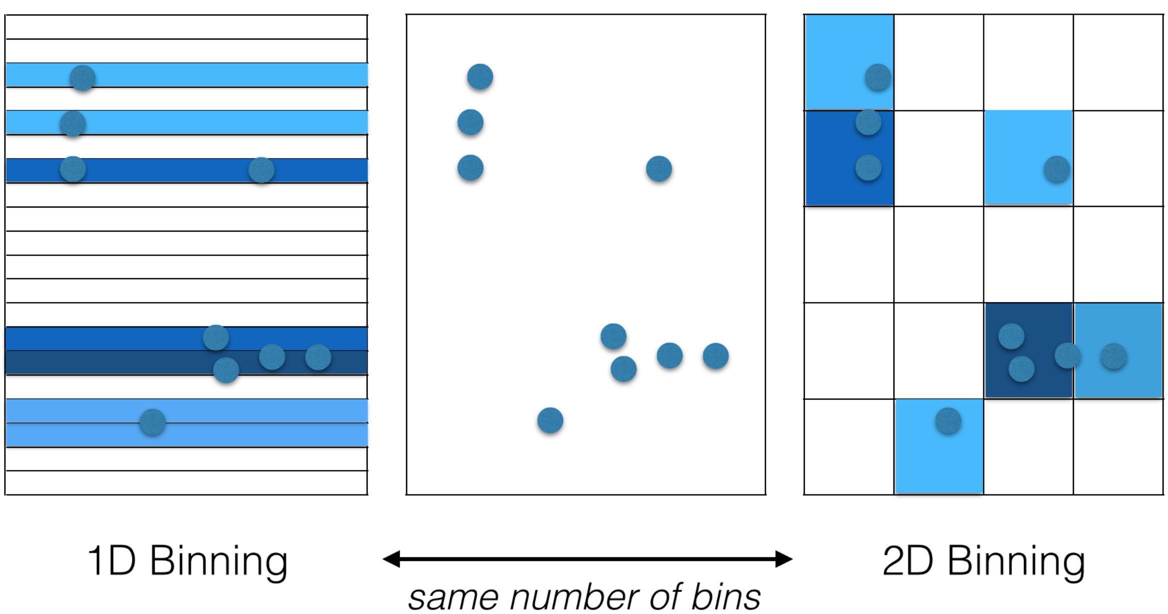
CONCEPTS

4.1 Residue Pair Transform Scoring

4.1.1 Motivation for Xform Scoring

Theoretical Motivation

The rosetta centroid energy, by reducing each residue to a single centroid point, reduces the relationship between residues to a one dimensional distance. Many centroid-style scoring methods, including some we have and are investigating, add in an additional angle term or two (like maybe the dot product between the two CA-CB vectors), and maybe a dihedral angle, giving a 2D-4D parametrization. However, the true relationship between two protein backbone reference frames (aka stubs, analogous to centroids) is inherently a 6D geometric transform. Yet nobody uses a 6D parametrization? Why the heck not? One reason is technical difficulty, but this can be overcome. The commonly cited “scientific” reason is statistics: “there just isn’t enough data to parametrize a 6D function.” That may or may not be true for more complicated models, but what we do 90% of the time is simple binning and I would argue that for binning, it’s best to use the full dimensionality of the data. Here are two possible representations of some made-up 2D data:



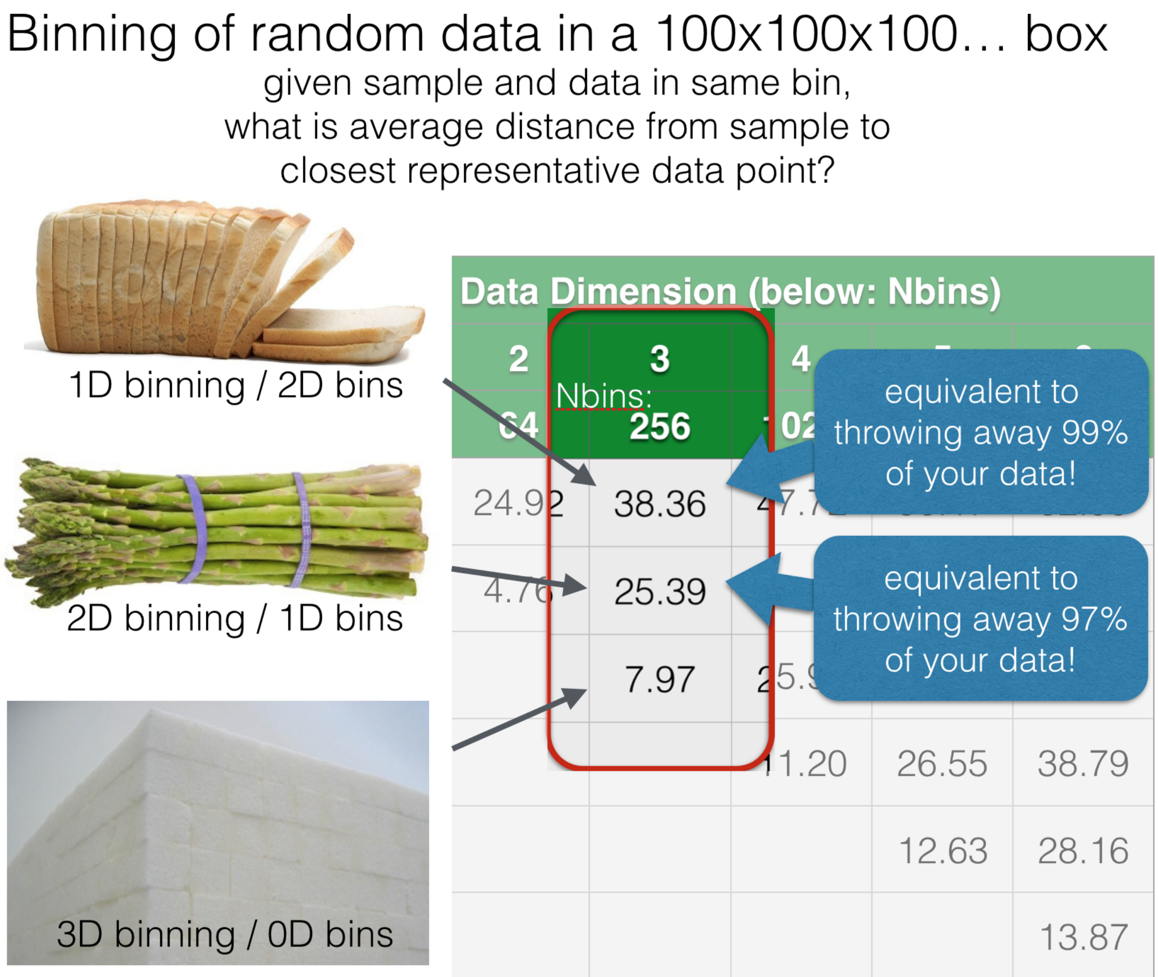
The one on the right looks much better to my eye. The key point here is that both representations use *the same number of bins*, so the statistical quality of the data is the same. This difference in quality becomes much more extreme as we look at higher dimensional data. Below is another totally artificial example looking at random data points in 100x100x100… boxes from 1D to 6D. In each case, we bin the data in 1..N dimensions, using the *same number of bins* and ask: for each binning strategy, what is the average cartesian distance from an arbitrary query point to the closest actual data point within the same bin?

Binning of random data in a 100x100x100… box

given sample and data in same bin,
what is average distance from sample to
closest representative data point?

		Data Dimension (below: Nbins)					
		1	2	3	4	5	6
	N:	16	64	256	1024	4096	16,384
Representation Dimension	1	1.57	24.92	38.36	47.72	56.17	62.99
	2		4.76	25.39	38.46	47.98	55.99
	3			7.97	25.93	38.31	48.32
	4				11.20	26.55	38.79
	5					12.63	28.16
	6						13.87

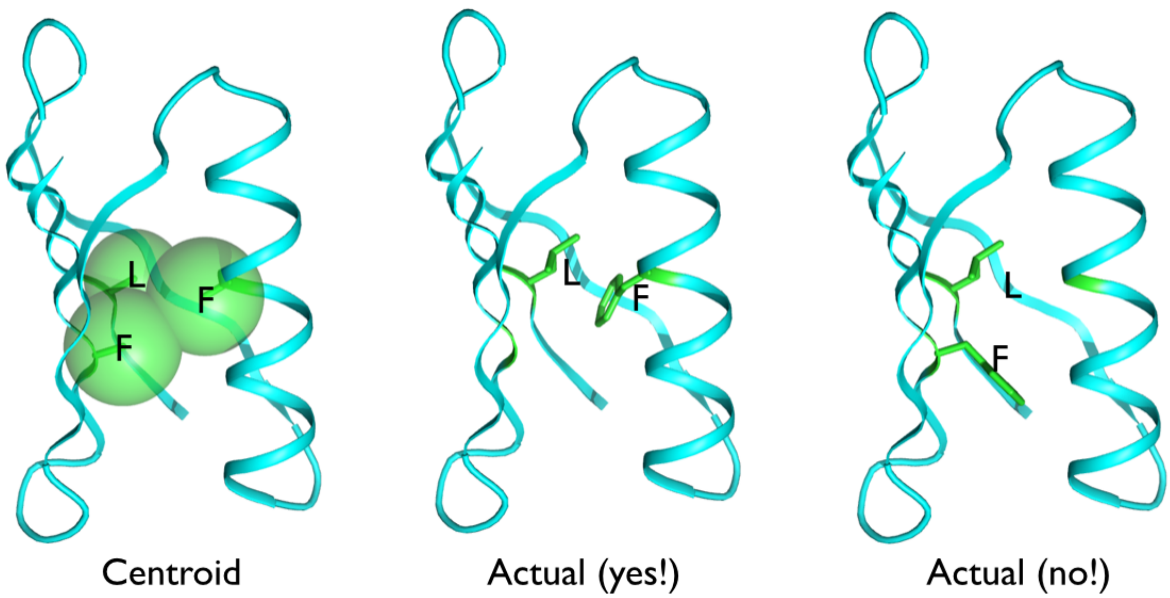
Here's an illustration for the 3D case:



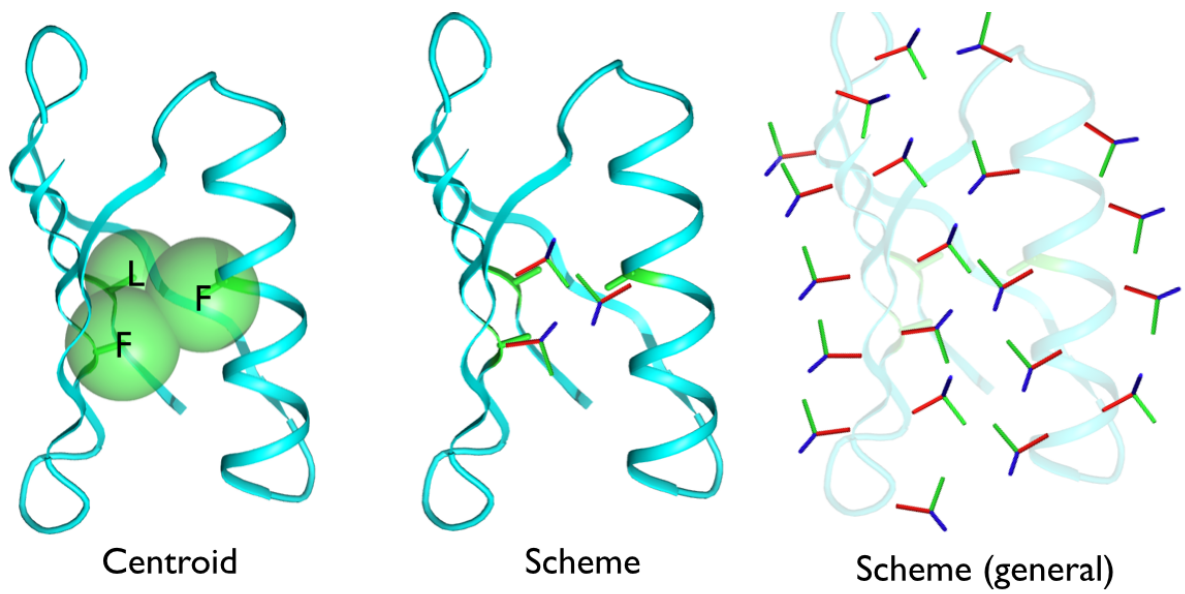
The Spherical Cow

Modeling complex objects as spheres (as we do for centroid) is the the subject of jokes:

Milk production at a dairy farm was low, so the farmer wrote to the local university, asking for help from academia. A multidisciplinary team of professors was assembled, headed by a theoretical physicist, and two weeks of intensive on-site investigation took place. The scholars then returned to the university, notebooks crammed with data, where the task of writing the report was left to the team leader. Shortly thereafter the physicist returned to the farm, saying to the farmer "I have the solution, but it only works in the case of spherical cows in a vacuum."



The above illustration of the core of protein G shows two LF residue pairs that look the same to the rosetta centroid energy because the centroids are almost exactly the same distance apart in both cases. But one is a highly favorable interaction with much contact area, while the other is a glancing interaction with little contact. There is *no way* to tell these cases apart with a 1D representation, the difference is all in the relative orientation of the two residues.



The middle panel above illustrates a representation of the residues with a full xyz coordinate frame. With such a 6D representation, these cases can be easily distinguished.

Accuracy

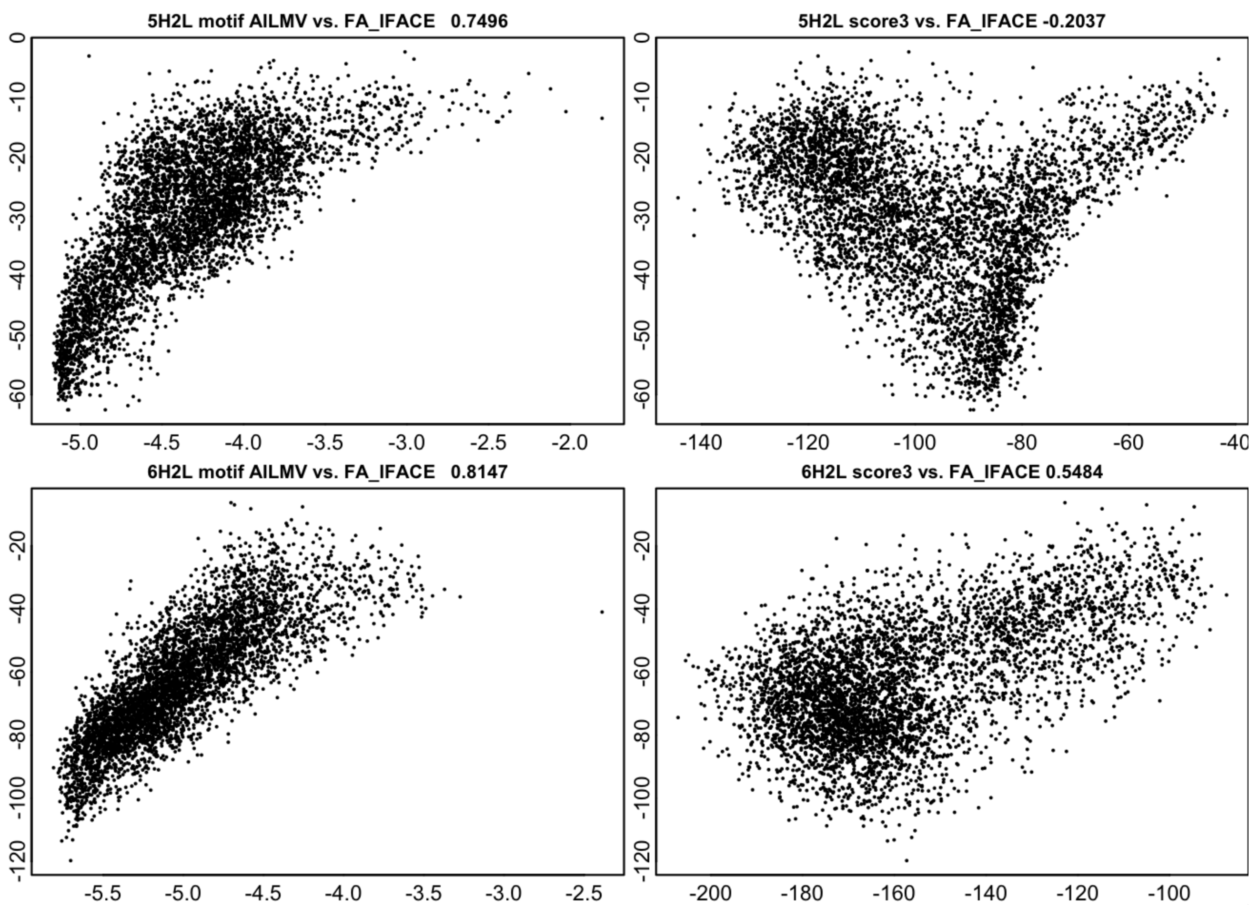
The transform based scheme score seems to be quite a bit more informative than any other "coarse grained" score I have seen. Docking prototypes based on this method seem to work very well for two sided interface design and reasonably well for some other tasks. We have not yet done much benchmarking, but hopefully will do more in the future.

The data below is for two fairly simple helical bundles of the Baker(TM) variety, one parallel one antiparallel. The plots below show Scheme and RosettaCentroid scores for backbone only structures *before* design plotted against Rosetta fullatom scores *after* a complete design process. Computing the Scheme score is approximately 10,000,000 times faster than the design calculation. These structures are close to a best-case scenario (because the interactions are helical pairs), but these are real backbones from a real project in the "wild", not an artificial test case.

Note: people argue about what centroid residue to use to represent a pre-design structure, some use VAL, some TYR, some ALA. Here, for the sake of an argument-free comparison, I allow the centroid score to "cheat" by putting the post-design sequence on the pose. This scheme score is *NOT* cheating here, but the centroid score is.

Note2: design here is done without any layer design, so this is somewhat artificial.

> Left panels: Scheme Score Right panels: Centroid Score X axis: Sheme / Centroid scores Y axis: Rosetta score post-design



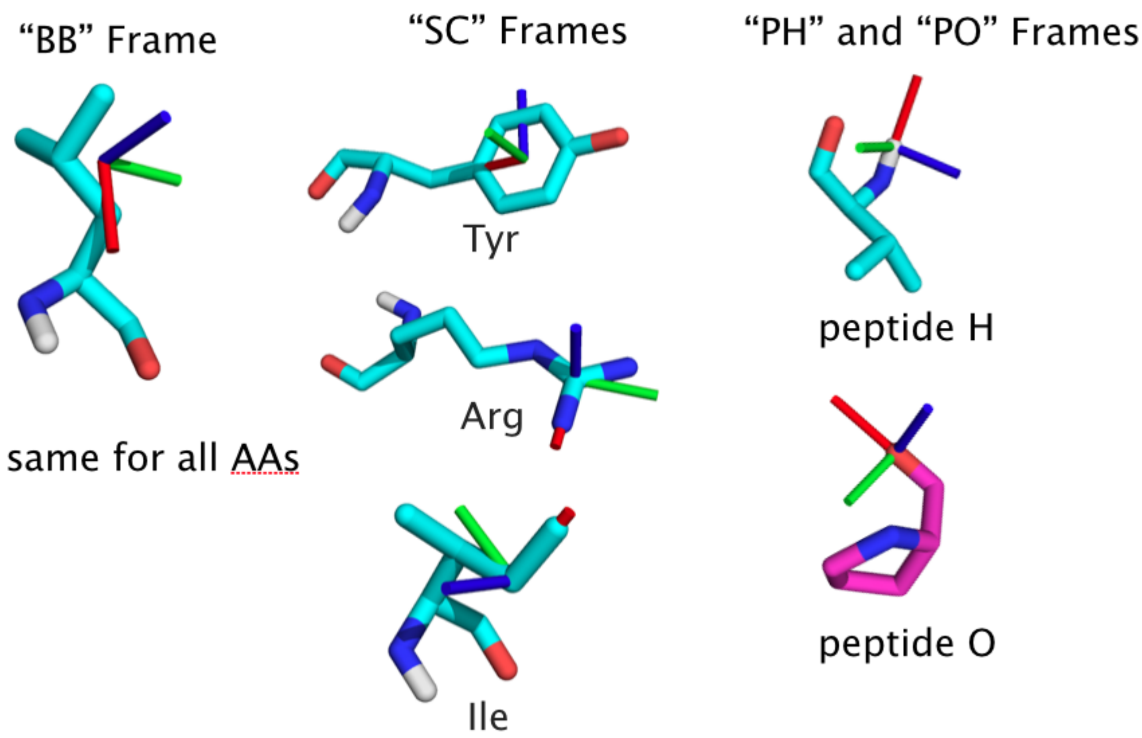
Flexibility

This is a highly general scoring model. The "scorable" elements of a body are made up of a coordinate frames, or *Actors*, which may represent any arbitrary functional group. By using different Petals and different ways of building score tables, we can apply Scheme to just about any protein modeling problem that involves searching conformation space (and maybe other problem domains too).

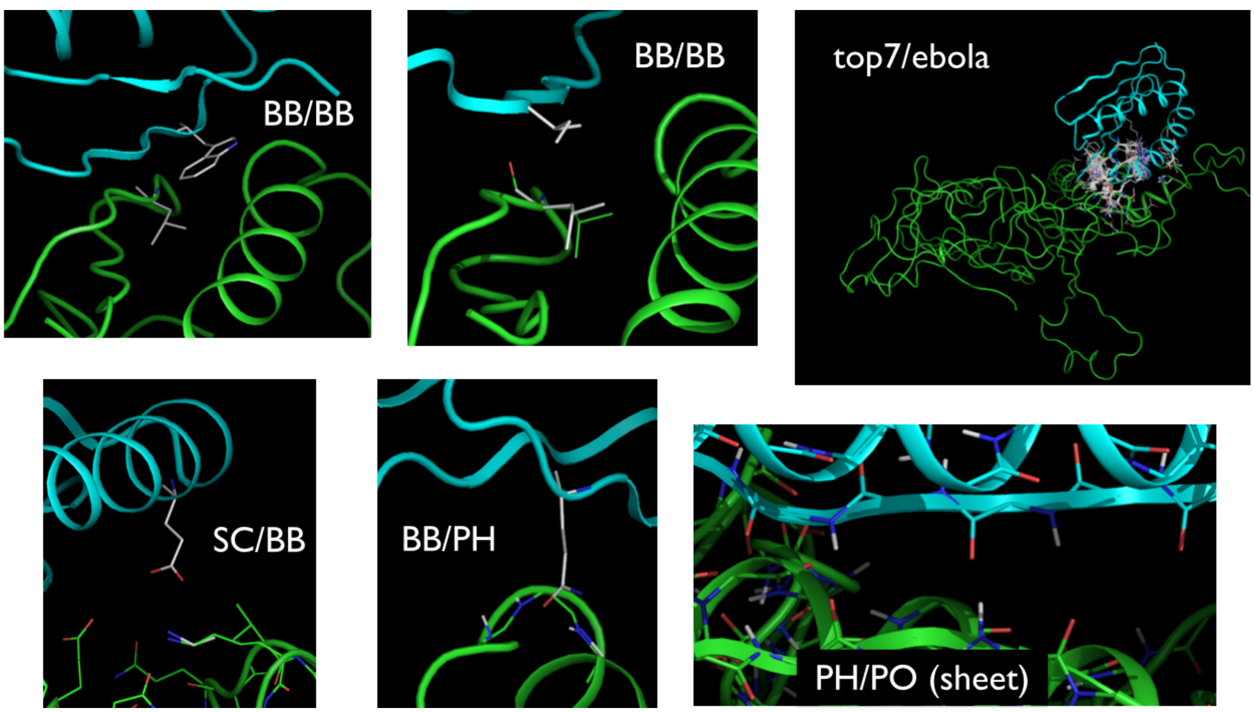
use case		Frame 1	Frame 2	Match SS	Match AA
centroid scoring		BB	BB	Yes	Yes
fullatom scoring		SC	SC	No	Yes
mixed cen/fa scoring		SC	BB	BB only	Yes
Loop scoring		BB Cterm	BB Nterm	Yes	Yes
one sided interface design		BB	BB	Res 2	Res 1
two sided interface design		BB	BB	Yes	No
"hotspot" & enzyme design		SC	BB	Res 2	Res 1
Interfaces w/ Beta Sheet Pairs		BB, edge strnd	BB, edge strnd	Yes	No
helix caps / edge strands		BB	BB	Yes	No
Ph Sensitivity via His-H-Nbb		BB, HIS Only	BB, N-H	No	No
Disulfides (metal sites, etc...)		BB, CYS Only	BB, CYS Only	Yes	No
linked construct design		BB Cterm	BB Nterm	Yes	No
domain swaps / insertions		BB Loop Stop	BB Loop Start	No	No

4.1.2 Actors

Score-able chemical entities, called Actors, are generally represented by full coordinate frames with a 3D position and orientation. The pair-score between two such entities is based on the full 6D transformation between the coordinate frames. Current Actor types are as follows. At some point, more general types will be available for hbond donors/acceptors (5dof ray), atoms (3dof xyz), and general parametric backbone systems.



Here is an example of motifs matching various actor pair types:



4.1.3 Challenges

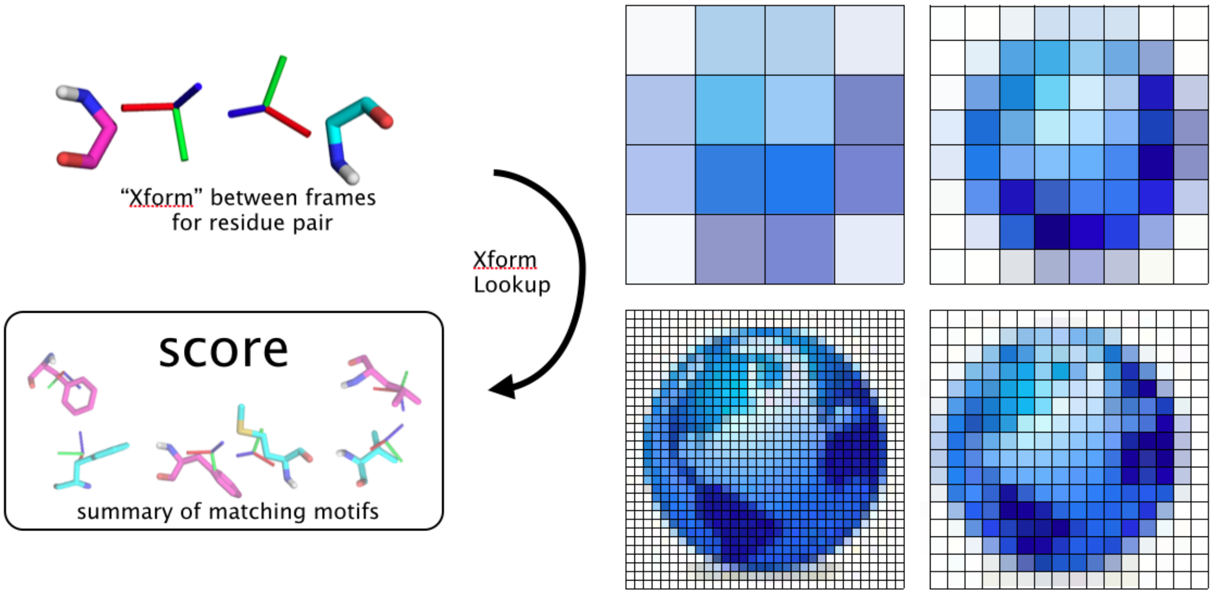
The 6D transformation space is very very large and score calculation presents some challenges The topic problem is how to score the rigid xform between a pair of "stubs" WLOG think of a pair of backbone N-Ca-C stubs and the possible side chain interactions they could allow. Native pdb motifs are pretty sparse in 6D xform space, which is a problem for polar interface design, loop design, any kind of one-sided design and even more so for structure prediction. We need more coverage to do well on these problems. Maybe we wait until the PDB is big enough, or maybe we replace or supplement with "denovo" score tables with Rosetta FA or some other continuous force field. In any case, the necessity will be much bigger score grids with trillions and trillions of data points going into them to reasonably cover the space. This is a technical challenge.

One part of the answer is hierarchal score grids; I'll give an example using the highest resolution we might reasonably use… ~2**64 = 1.8e19 cells which gets us a resolution of 0.016A translation and 0.3° rotation (more than sufficient for sidechain interactions.. I don't see any need for > 64bit indices in scoring). Call this G8. Obviously we can't store, or even compute, everything in such a huge grid… It will be sparsely populated with only the very best and/or most geometrically specific (steepest gradient) interactions. If a data point (or more likely a block of 2**6 or 4**6 data points) don't make the cut for the finest grid, it could go in the next finest grid G7, its parent, which will have ~2**58 cells at 0.032A/0.6° resolution. Each cell in the parent exactly covers 2**6 = 64 child cells, hense the 6 bit reduction in size. If not there, it could go in the next finest G6 with ~2**52 cells, or the next G5 ~2**46, and so forth to the base grid G0 with ~2**16 cells and a resolution of maybe 4.0A and about a radian. So in this instance there would be a nested hierarchy G8-G0 with 9 levels.

When looking up the score for an interaction, you must figure out the highest G which actually contains your point, then look up its score. Naively, this could require querying up to half the grids on "average". Using some kind of skewed binary search maybe only 2 or 3. But we'd really like to make only one grid lookup because each one is expensive. This for two reasons: (1) the mapping from an RB xform to an index number is nontrivial, and (2) looking up the resulting index in memory via a hash table or whatever takes a while because there is probably a cache miss involved. The nested grid/indexing setup I've just finished mitigates (1) by allowing all grids G0-G8 to use basically the same index. For (2), instead of checking each one, or doing a binary search of something, a bloom filter for each resolution grid can tell you 99 times out of 100 whether the data you want is available using only maybe 65K or so of memory. (it's an interesting data structure… worth checking out)

Hierarchical Score Grids

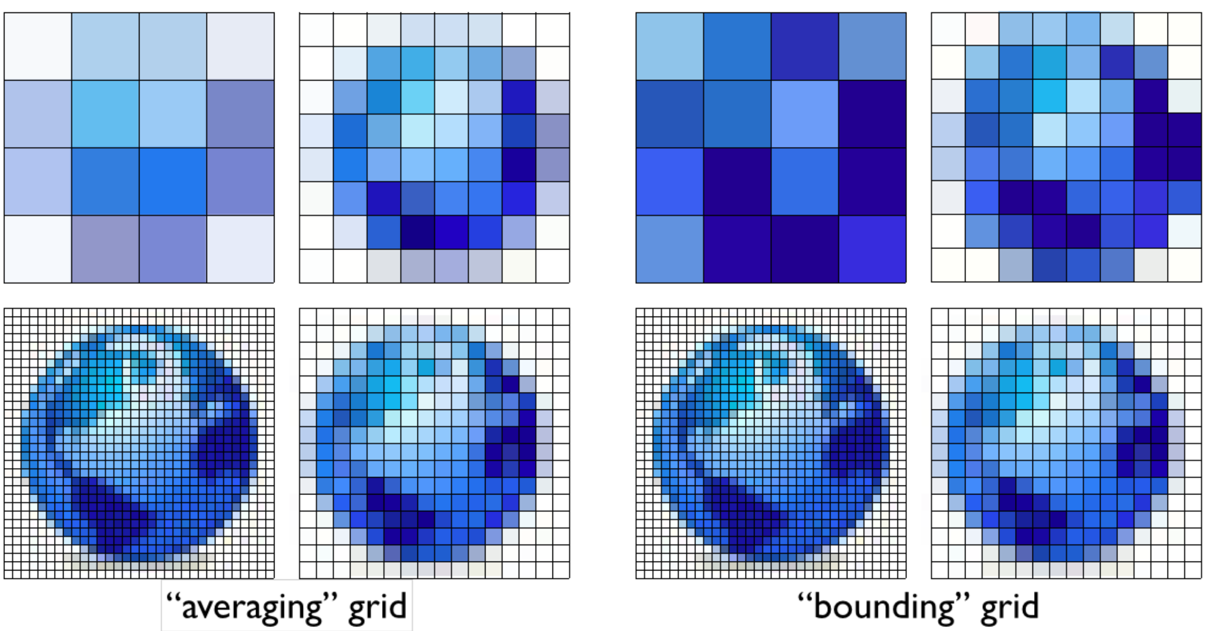
Scores must be computed such that they evaluate an ensemble of conformations, not just a single one. Sampling is done in a hierarchy such that each sample point must cover a defined region of space. High up in the hierarchy, a sample point would be responsible for a large volume of space, say, 5Å in diameter with orientational deviation up to 15°. Lower down in the iterative sampling hierarchy, a sample point might represent a region 0.1Å in diameter and 1° orientational deviation.



Bounding Score Grids

This, along with hierarchical decomposition, will allow implementations of Branch and Bound searches.

It is possible to construct score functions that rigorously bound the best possible score in a region of conformation space. If scoring can be set up in this manner, along with careful tracking of the volume sample points must represent, a branch and bound search is possible. Such a search would guarantee that no possible solution worse than a reported threshold was missed in the search.



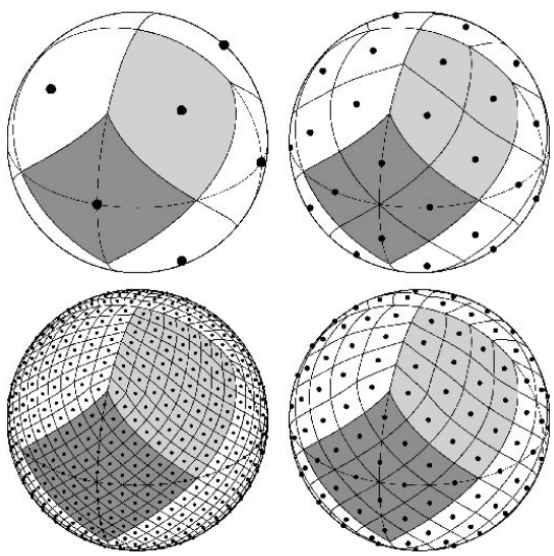
4.2 Hierarchical Sampling and Scoring

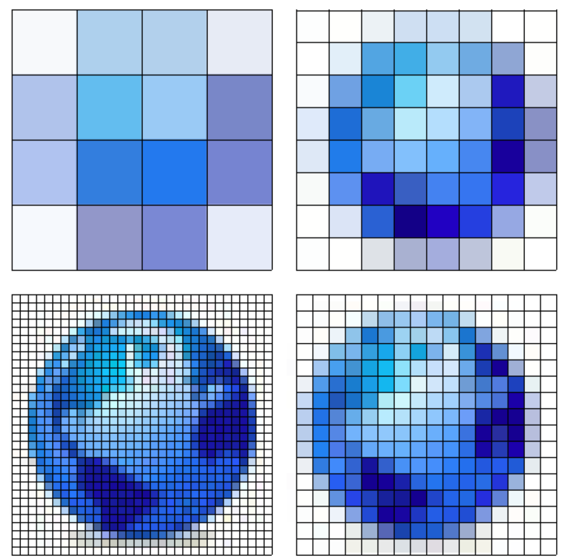
4.2.1 Hierarchical Search

In SCHEME all DOFs are indexed in a nested hierarchy. The number of samples along each dimension doubles with each increase in resolution in a nested manner (see illustrations). This nesting forrest of conformations can be searched efficiently by first examining the lowest resolution points with a correspondingly low resolution objective function, then choosing a best-scoring subset for further analysis. All the children of this subset are then examined with a higher resolution objective function, and another best-scoring subset is generated. This process repeats to the desired precision, probably 5-10 iterations for most protein sampling problems.

Below is an illustration of a nesting hierarchical search grid (from Healpix):

Example:





Sampling Everything

Say we search a 6 dimensional DOF space, a simple asymmetric docking problem, and the coarsest reasonable resolution is 4 samples per dimension, or 4,096 points, corresponding to a sample roughly every 8Å. The total number of samples to evaluate grows as the 6th power of the number of samples per dimension:

- Number of samples at resolution 8.0Å is 4,096
- Number of samples at resolution 4.0Å is 262,144
- Number of samples at resolution 2.0Å is 16,777,216
- Number of samples at resolution 1.0Å is 1,073,741,824
- Number of samples at resolution 0.5Å is 68,719,476,736
- Number of samples at resolution 0.25Å is 4,398,046,511,104
- Number of samples at resolution 0.125Å is 281,474,976,710,656

Sampling Hierarchically

obviously, simple enumerative sampling is impossible at a reasonable resolution, even in this simple case of two body docking. If instead we adopt an iterative hierarchical search in which we prune away areas of the search space which are *not likely to contain good solutions*, the number of samples which must be evaluated no longer grows exponentially. Say at each stage we keep only the best 10,000 samples, and evaluate all their "children" in the next stage. In this 6 dimensional case, each sample has 64 children so we evaluate at most 640,000 points at each stage:

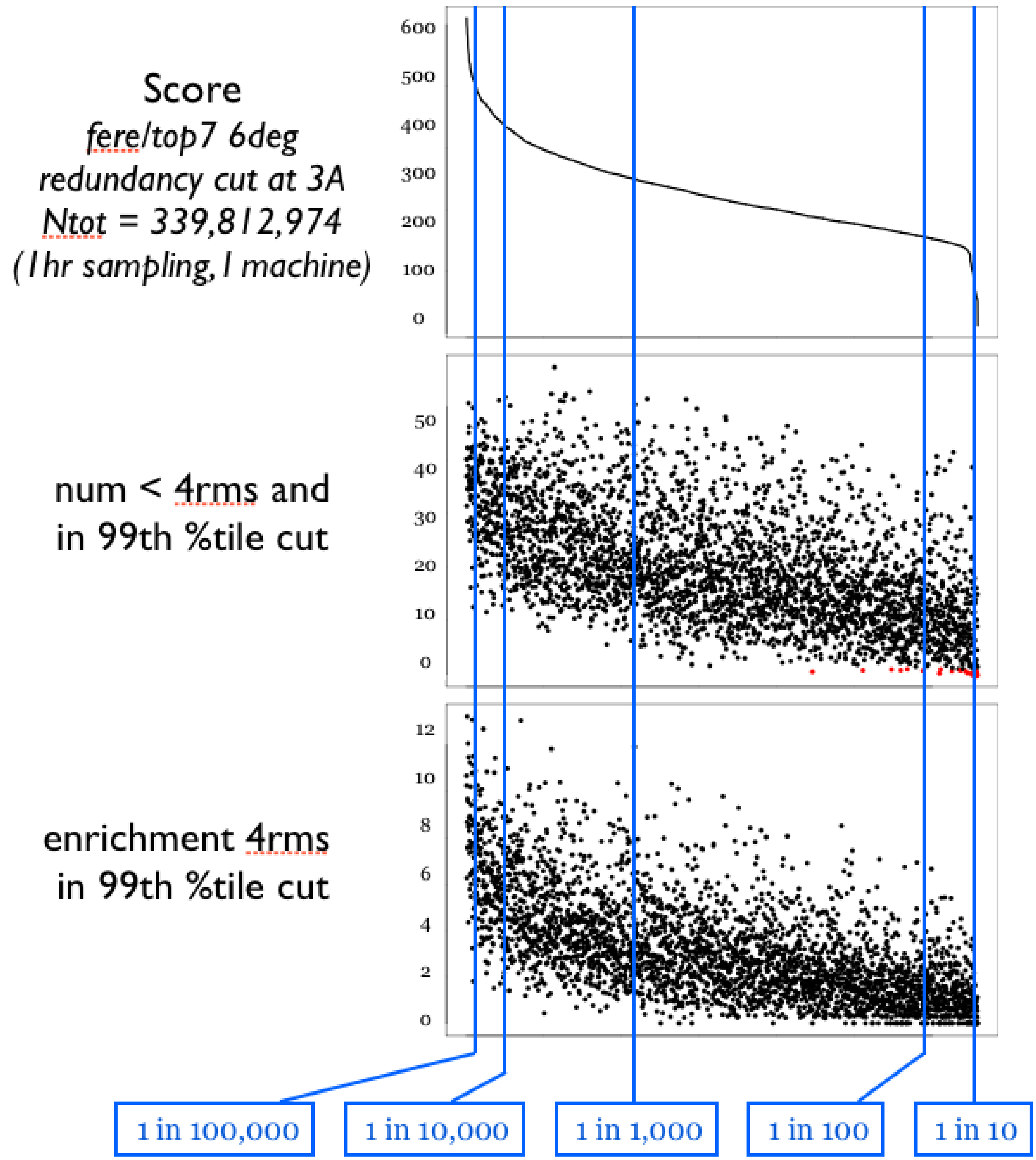
- Number of samples at resolution 8.0Å is 4,096
- Number of samples at resolution 4.0Å is 262,144
- Number of samples at resolution 2.0Å is 640,000
- Number of samples at resolution 1.0Å is 640,000
- Number of samples at resolution 0.5Å is 640,000
- Number of samples at resolution 0.25Å is 640,000
- Number of samples at resolution 0.125Å is 640,000

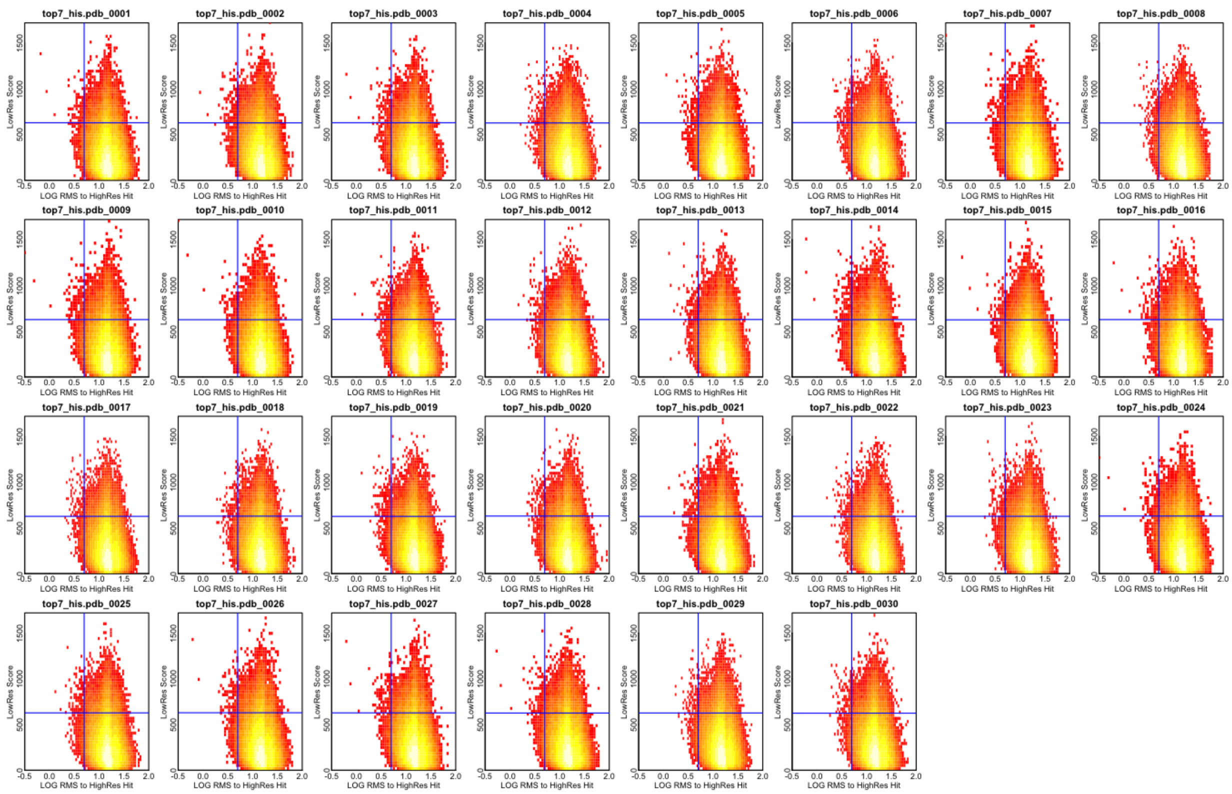
So the total number of samples is 3,466,240. Scheme scoring is quite fast, on a 4-core cpu in the current implementation, this would take about 30 seconds.

Is Hierarchical Search Really Global Search?

Of course, we may miss some solutions by discarding all but the top 10,000 at each stage. In some simple empirical tests, is seems that we don't miss too much. The figure below shows a comparison between two sampling runs, one sampling roughly every 0.8Å with ~340 million samples, and one sampling roughly every 4.0Å with ~100,000 samples. The top panel shows a *redundancy filtered* distribution of scores for the large sampling run. Because we want to examine how "deep" into the global landscape of the larger set we can "see" based on the smaller set, the large set is pruned very aggressively on redundancy, no structures within the set are within 3Å rmsd of each other. The X axis on all three plots corresponds to the sorted index of structures in this high-res, redundancy-filtered reference set. The middle panel plots, for each structure S in the reference set, the number of structures in the low-resolution set which are [1] in the top 1/100th of the set by low-res score, and thus would be candidates for more refined sampling, and [2] are within 4Å rmsd the reference structure S, and thus "cover" S and could recover it in a search. It seems from this data that the hierarchical search approach should not prune away too many "good" candidates.

This test was done with "averaging" type scoring, NOT the superior *bounding* type scoring.


Score
fere/top7 6deg
redundancy cut at 3A
Ntot = 339,812,974
(1hr sampling, 1 machine)
600
500
400
300
200
100
0
num < 4rms and
in 99th %tile cut
50
40
30
20
10
0
enrichment 4rms
in 99th %tile cut
12
10
8
6
4
2
0
1 in 100,000
1 in 10,000
1 in 1,000
1 in 100
1 in 10



4.2.2 clash checking

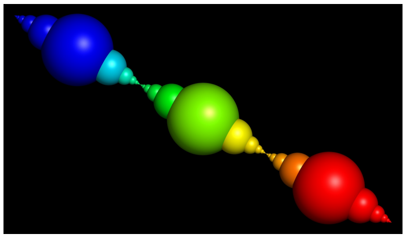
We can totally solve the hierarchical clash checking problem using an euclidean distance transform (EDT) on the excluded volume elements (I say we call them "Thorns"). This will give, for each atom, a depth below the surface of the molecule. If you're clash-checking at a resolution of, say, 2A, you simply set the atomic radius at (depth-2A), ignore if below zero, and your good. Will be easy to use the existing clash-check machinery with this extension. Also gives the opportunity for very fast approximate clash checking by picking some small representative set of atoms with high depths – the depth can be used as a radius for clash-checking, so atoms with high depths can exclude alot of volume with only a handful. Obviously, picking the representatives will take some care, but I think we'll get alot of speedup out of a fast non-strict clash check even if it isn't prefect.

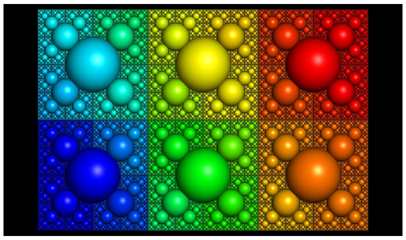
4.2.3 Nesting Enumerated State Tree (Nest)

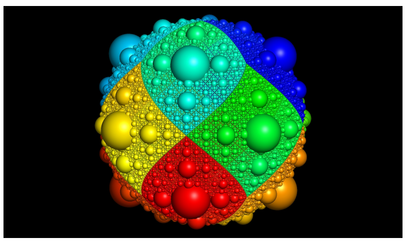
All DOFs are managed by Nest objects, which define a hierarchical tree of all allowed states. Nests can control: - Stem transforms in the Bouquet (scene tree) - Rose conformation DOFs (clustered fragments, parametric backbones, etc) - Helical symmetry DOFs (helix and screw axes)

Illustrations

some misc. illustrations showing the nesting grids:



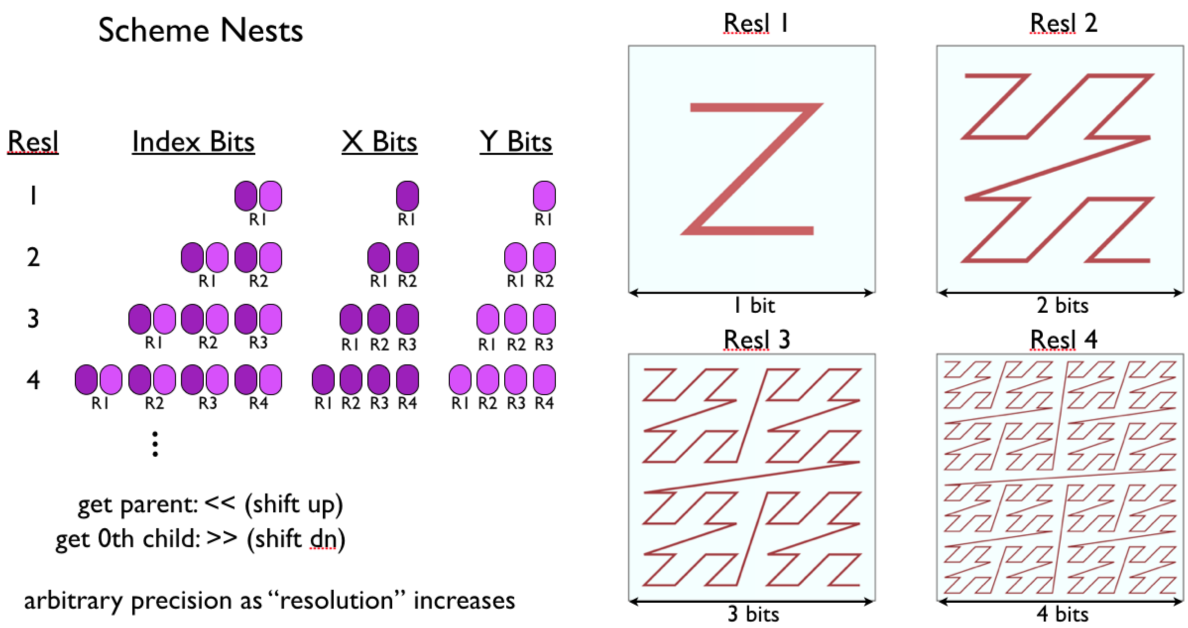




4.2.4 Indexing Info Partially Obsolete!

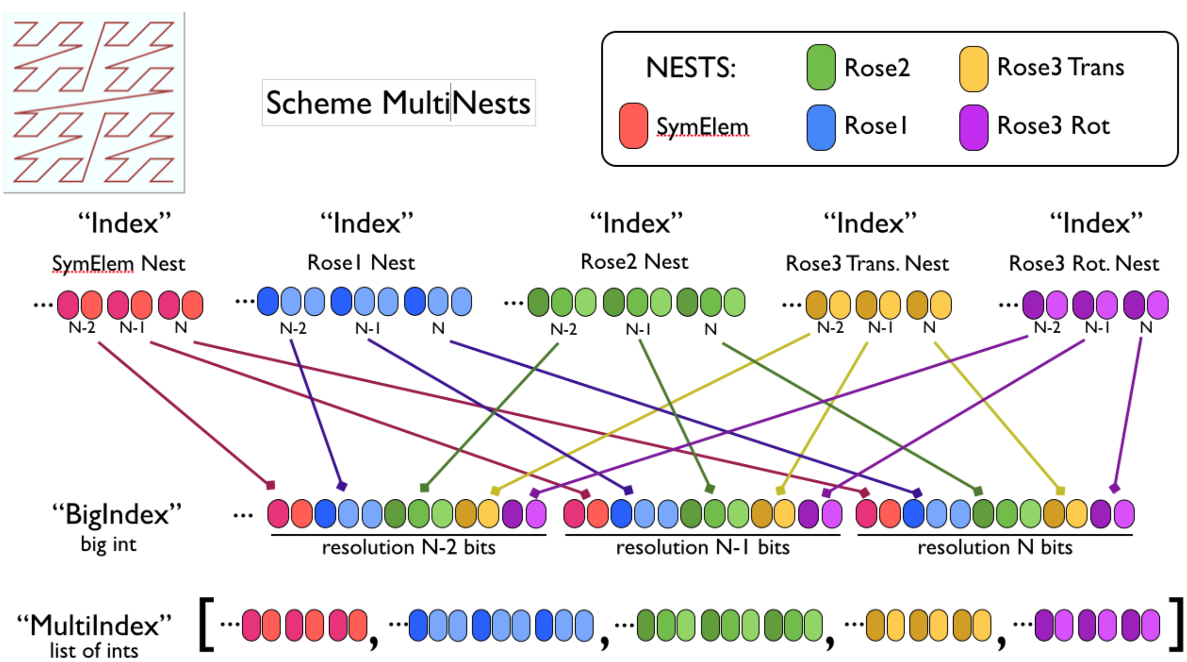
Zorder Indexing

Illustration of basic Nest indexing, which using "z-order" indexing and "morton numbers" in order to (1) make sure siblings in the hierarchy have contiguous indices, avoiding explicit storage and listing, and (2) increase memory coherency by ensuring similar conformations are sampled together. Simple bitshifts are all that's needed to get the parent or children of an index.



MultiNest Indexing

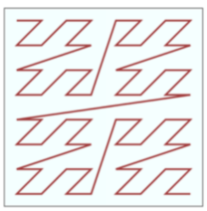
Illustration of a Scheme MultiNest index structure, showing the "BigIndex" strategy which preserves zorder indexing for a set of Nests, and the "MultiIndex" strategy which just lists individual indexes. We may need to employ some mix of these constructs to cope with the indexing headaches involved with *Hierarchical Packing*.

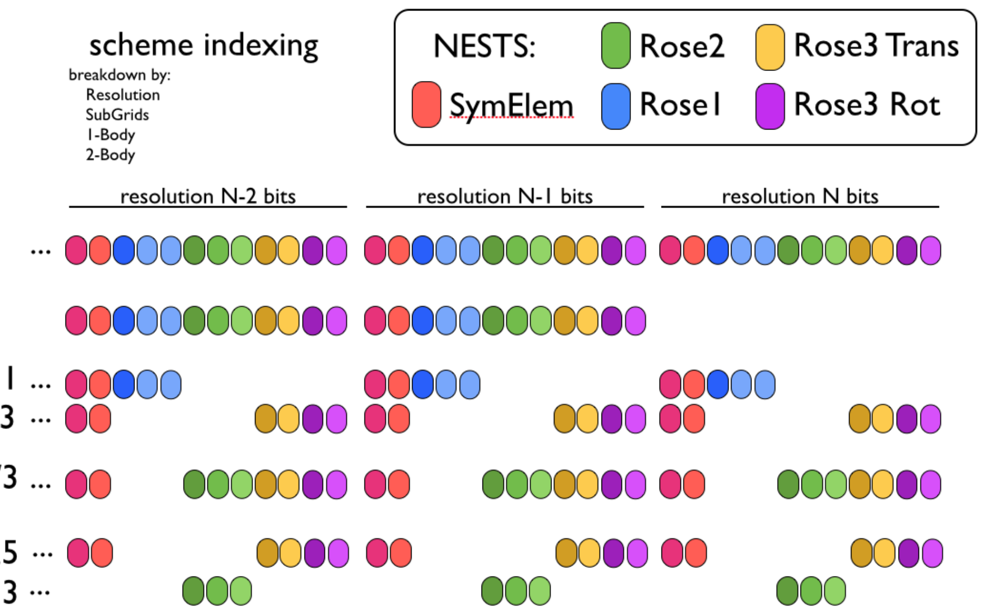


Indexing Challenges

- must be able to sort and iterate based by Zorder.
- must be able to decompose by subsets of Nests within MultiNest (for pair-decomposition)
- must be able to decompose by individual Nests
- must be able to efficiently represent a list of indices to be evaluated in a Course of sampling
- BigIndex with arbitrary precision implementation is terrible, each on the heap
- MultiIndex maybe better, if done with 2D array, but how to increment & sort efficiently
- Preferably, list only the "parents" at a given level in the hierarchy, as all children of each parent will be checked

Here's an illustration of what some of these different indexing types are





I propose some kind of index manager structure that can assist with all this. It's implementation can at first be simple and slow, and later be replaced with something more nastily efficient with the same interface. Something like this:

- class MultiIndexSet
  - **-Types**
    - strong_typedef Size IndexMS // sample number in current Course
    - strong_typedef Size IndexG // Index of Nest (to select a nest)
    - strong_typedef Size IndexRB // index of Rose
    - strong_typedef Size Index1B // Index of one-body states
    - strong_typedef Size Index2B // Index of two-body states
  - **-Attributes**
    - explicit_parent_indices : BigIndex
  - **-Operations**
    - IndexMS nstate_total () // total number of samples to be done in set
    - Index nstate_grid (IndexG) // size() of one Nest

- Void index_grid (IndexG, IndexMS) // Nest index
- Index1B nstate_onebody (IndexRB) // number of distinct one-body states in set for a body
- Index1B index_onebody (IndexRB, IndexMS)
- Index2B nstate_twobody (IndexRB, IndexRB)
- Index2B index_twobody (IndexRB, IndexRB, IndexMS)

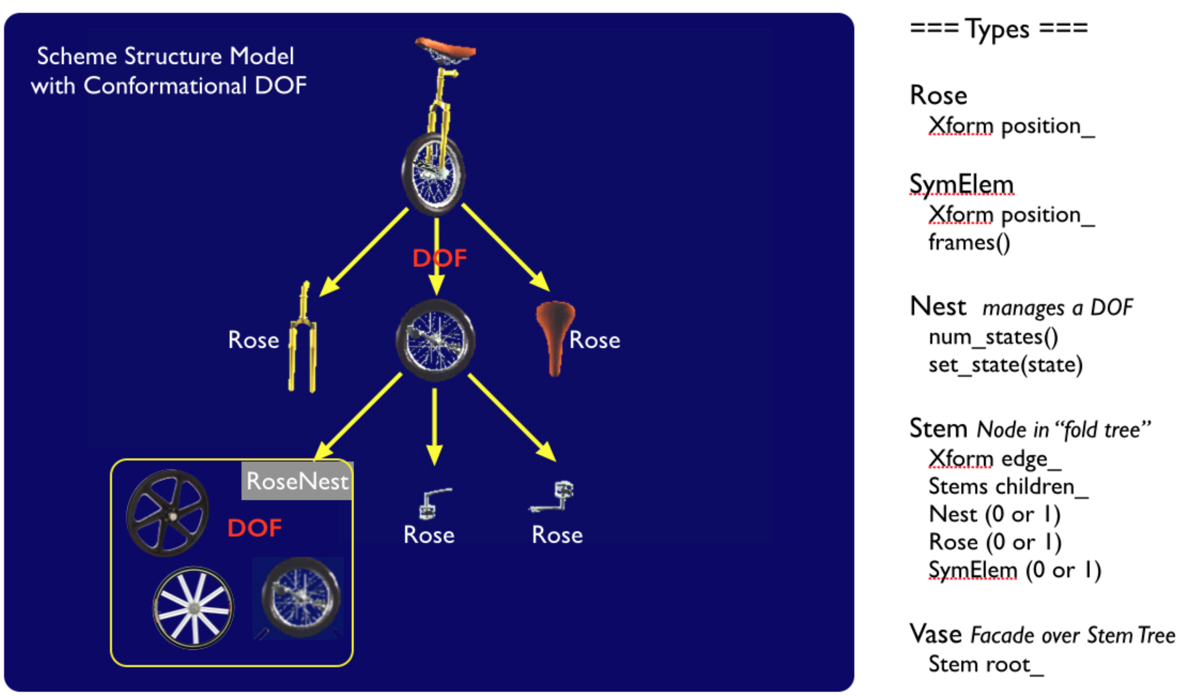
4.3 Rigid Body Representations

this is out of date, concepts are similar, but constructs in rpxdock are named and behave differently

Basic element of structure is a “rigid” chunk of protein structure called a Rose. This “rigid” structure may actually be a family of related conformations, if a proper hierarchical decomposition of conformations can be made.

Kinematic model is Roses and SymElems placed by a “scene tree” type structure called a Bouquet. All DOFs, including transforms which place entities in the tree, helical symmetry DOFs, and Rose conformations, are managed by hierarchal nested grids called *Nests*. Here is a silly example:

Scheme Bouquet Illustration



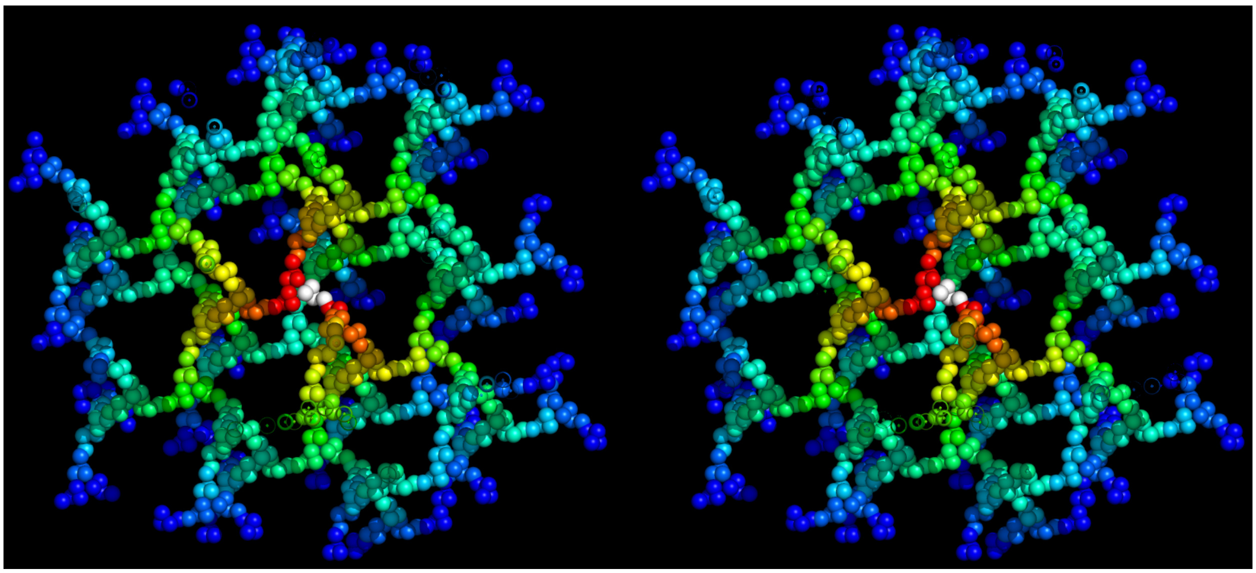
Here is the proposed object structure, which provides [1] efficient representation of multiple identical bodies in different positions (symmetry, multiple threads, etc…), [2] Interactions indirected WRT rigid body position and conformation choice.

4.4 Dynamic Symmetry

Symmetry is represented by explicit symmetry elements which are placed in space in the same way that physical entities are. Symmetrically related coordinate frames are dynamically generated based on the placement of symmetry elements by (1) using a depth first search up to a specified depth to generate a symmetric topology and (2) each time the state of the system is changed, positions of the symmetric frames are updated based on the new symmetry element positions. Using a Trie based data structure to represent the symmetric topology makes this dynamic approach efficient enough that it is not a bottleneck, even for very complex crystal symmetries that require a deep search.

Below is a walleye stereo image of an I23 symmetry, colored by depth in the symmetry update trie. The trie-based update mechanism seems to be efficient enough to allow dynamically updating such deep symmetries as below without noticeable overhead.

I23 symmetry colored by SymUpdateTrie depth



4.5 Pair Decomposition (Packing) Ideas

All scores should be pair-decomposable. In combination with the gridded ridged body kinematic model, this allows any system with more than two rigid bodies to be optimized by precomputing pair-energy tables. By analogy to the packer, each rigid body is like a "residue" and each sampled position of a rigid body is like a "rotamer."

Pair decomposition of score calculations is a key concept for combinatorial optimization of protein structure. It is the basis for the Rosetta packer, and other protein design methods such as DEE/A*. The core concept is that no matter how many degrees of freedom there are in a system of independently changing bodies, the energetics of the system can be captured (if the energy function is pair-decomposable) entirely by the pair interactions only. Of course these are coupled, but it does mean that score calculations can be done once to produce one and two body score tables. Then optimization can be carried out entirely with lookups into the precomputed score tables.

This notion is highly complimentary with hierarchical search. At each stage of the search hierarchy, a pair-decomposition and optimization procedure is applied to the states listed for evaluation in that phase of the hierarchy.

The pair-decomposition of energies, which is the basis for "packer" type algorithms, should be universally applicable for faster energy calculations in all systems with more than two bodies (or two + symmetry), provided sampling is all on a predefined grid. So I'm thinking this should be build into some kind of universal energy caching data structure. (It would even be of use in folding-type kinematics, you could al least have your N to N+1 and N to N+2 interactions precomputed and never have to repeat them. And/or do some type of annealing/DEE on the local interactions and see what you get globally… branch and bound won't be possible, but it may still work well) To do this requires adding a little more facility to the CompositeGrid and kinematics to extract the degrees of freedom that place individual bodies and pairs of bodies, but it should be possible to do it in a general manner. Sorting out the indexing will likely be a headache.

4.5.1 Hierarchical Packing

Illustration of hierarchical packing, one cycle.

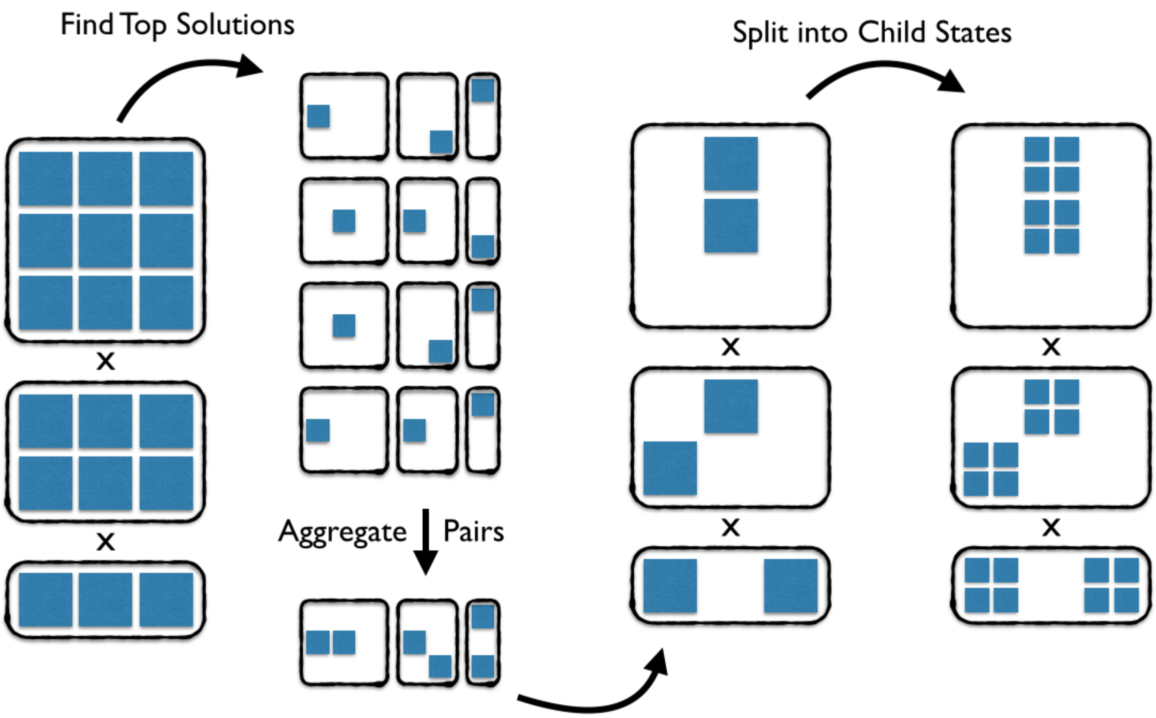
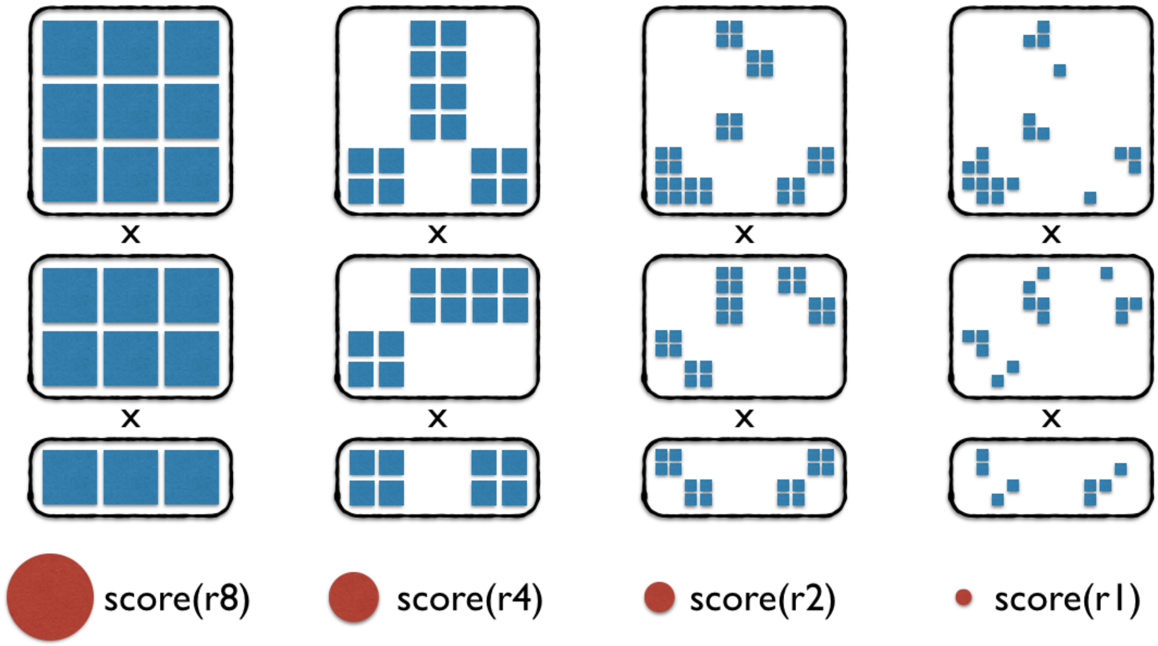


Illustration of considered states over four round of hierarchical packing. Red circles: Covering radius (i.e. radius or convergence, resolution) of the score function used in a given round… must be be matched to the sampling resolution.



Rosetta Analogy

In rosetta design, we generally include "extra" rotamers when packing. These extra rotamers are generated by taking the base rotamers and tweaking them a little, say by adding ± 1 standard deviation to each chi angle. These extras are all added into the rotamer pool and treated uniformly, ignoring their relationship to the "parent" rotamer. In practice, we generally do exactly this, with 3**Nchi total rotamers, because it improves the designs that come out and the extra work isn't too much. But what if you want to sample rotamers more finely? One of the extra rotamer modes in rosetta is ± 0.5, 1.0 and 1.5 sd, yielding 7**Nchi rotamers per residue. This produces a "rotamer explosion" making the packing problem too big to solve in a reasonable amount of time/memory.

If you wanted to take advantage of these closely-related rotamers to reduce the calculation, you could do something like this:

- Pack with no extra rotamers using a very soft score function.
- take the top 10,000 solutions produced by the packer and for only the rotamers involved in those top 10,000 solutions, add +-1 SD rotamers.
- now repack with this expanded-subset of rotmers with a lightly soft energy function.
- Again take the top 10,000 solutions. Make a new rotamer set with only rotamers involved in those solutions, and add in +-0.5 SD rotamers.
- now repack with the standard hard energy function.

Hopefully this gives some feeling for how hierarchical packing will work in scheme. I'm not sure how well this would work without carefully "softening" the energy function in Rosetta. In scheme score functions of the appropriate "covering radius" can easily be generated. Probably best with bounding energies.

Scheme Hierarchical Packing Use Case

Write me!

4.5.2 Challenges

Packing Algorithm

We need a "Packer" that can emit a large set of top solutions efficiently. DEE would be nice for optimality, but it may be too slow.

Bollox to DEE and clever algorithms, Monte Carlo with some quenching and a small taboo list seems to work great. Try a billion substitutions, keep the top million things you see, done in 60 seconds.

4.6 Concepts Summary

4.6.1 Ridged body structure model

A molecular system is broken down into some number of rigid bodies. Depending on the system being modeled, these rigid units could be individual atoms, residues, fragments, whole chains or even assemblies of chains. These elements are placed in space using a transform based fold tree (like a scene graph). *[More Info]*

4.6.2 Dynamic Symmetry Model

Symmetry is represented by explicit symmetry elements which are placed in space in the same way that physical entities are. Symmetrically related coordinate frames are dynamically generated based on the placement of symmetry elements. *[More Info]*

4.6.3 RB Transform Based Scoring

Chemical entities are represented by full coordinate frames with a 3D position and orientation. The pair-score between two such entities is based on the full 6D transformation between the coordinate frames. *[More Info]*

4.6.4 Hierarchical Representations

All sampling is done on an arbitrary precision, hierarchically nested grid. This allows iterative refinement in search using techniques like branch & bound. *[More Info]*

4.6.5 Pair Decomposition

All scores should be pair-decomposable. In combination with the gridded ridged body kinematic model, this allows any system with more than two rigid bodies to be optimized by precomputing pair-energy tables. By analogy to the packer, each rigid body is like a "residue" and each sampled position of a rigid body is like a "rotamer." *[More Info]*

4.6.6 Ensemble Energy Evaluation

Scores must be computed such that they evaluate an ensemble of conformations, not just a single one. Sampling is done in a hierarchy such that each sample point must cover a defined region of space. High up in the hierarchy, a sample point would be responsible for a large volume of space, say, 5Å in diameter with orientational deviation up to 15°. Lower down in the iterative sampling hierarchy, a sample point might represent a region 0.1Å in diameter and 1° orientational deviation. *[More Info]*

4.6.7 Branch and Bound

It is possible to construct score functions that rigorously bound the best possible score in a region of conformation space. If scoring can be set up in this manner, along with careful tracking of the volume sample points must represent, a branch and bound search is possible. Such a search would guarantee that no possible solution worse than a reported threshold was missed in the search. *[More Info]*

CHAPTER

FIVE

APPLICATIONS

5.1 Fan contributions

this is where you cage docking folks can exhort rpxdock for. . . . doin' stuff

5.2 Old Stuff

5.2.1 Applications

TODO: need current applications of actual rpxdock and maybe rifdock The concepts described here also apply to the rifdock project. The MatDes docking methodology, sicdock, "Motif Designer" and "Motif Docking" in use in the Baker Lab amount to early prototypes of rpxdock.

Two Sided Design (easy)

Cage Design

Neil, Jacob, Yang Very early versions used in the Science and Nature papers, "motif docking" used in latest icosahedron set and Jacob seems to think it was an improvement.

Cyclic Oligomer Design

Jorge: seems to be working fairly well, and with a very limited scaffold set.

Jeremy: 1 of 3 designs seems to work very well

Lattice Design

Ariel, soon to go to science or nature

Helical bundle design

used for scoring

Possu, Gustav

Repeat Protein Design

used for scoring

TJ et al?

Heterodimer Design

Daniel says it's fantastic and many of his hetero-dimer designs may work. we'll see if they're specific.

One-sided Binder design (hard)

RIFdock

works well, see Jiayi's HBI binder paper in nature, all the IPD work on protien and small molecule binders in the past 2-3 years is rifdock??

MotifDock

needs replacing with a rpxdock protocol!!

Aaron says it works well and is using it. Luke, Kenwuan and David La have used it and said it was helpful, but I dunno if they're still using it.

Structure Prediction (0-sided, very hard)

no data, need to do benchmarks!

CHAPTER

SIX

TODO

6.1 Simple-ish Todos

- only certain residues

6.2 Needed Frameworks

6.2.1 Scoring Framework

For use in array-ified main docking

- filter
- trim
- score
- examples
- termini orientation
- intra interface

6.2.2 Post doc filter framework

For use after main dock, filtering one dock at a time and only a limited number of results. Theses filters can be much more flexible and probably do arbitrary calculations

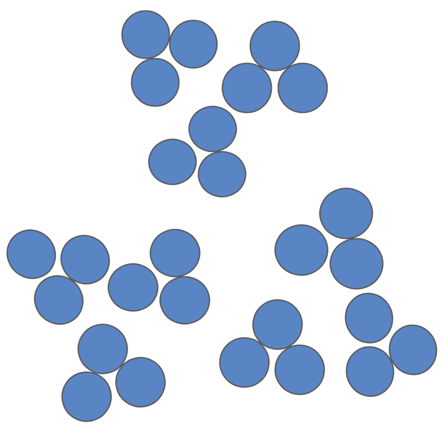
- ss count??? (maybe in filtering if efficient)

6.3 Documentation

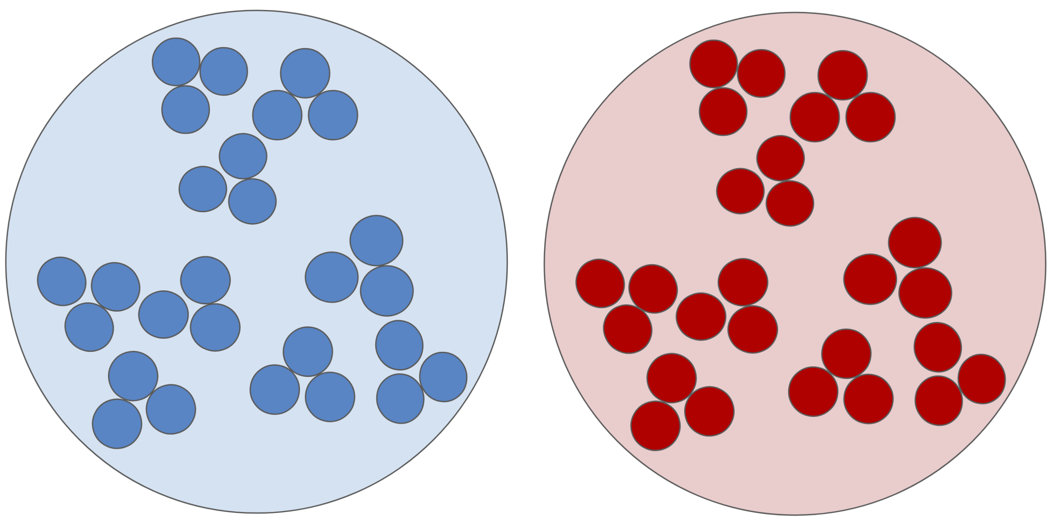
need much much more documentation, both tutorial stuff and module (code comment) docs

6.3.1 Figures to doc and place

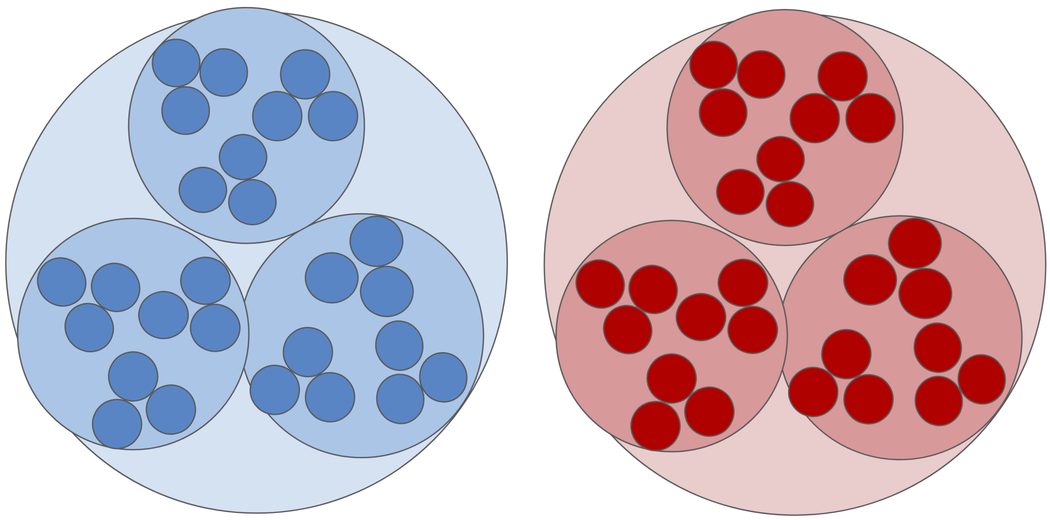
GM_bvh_1



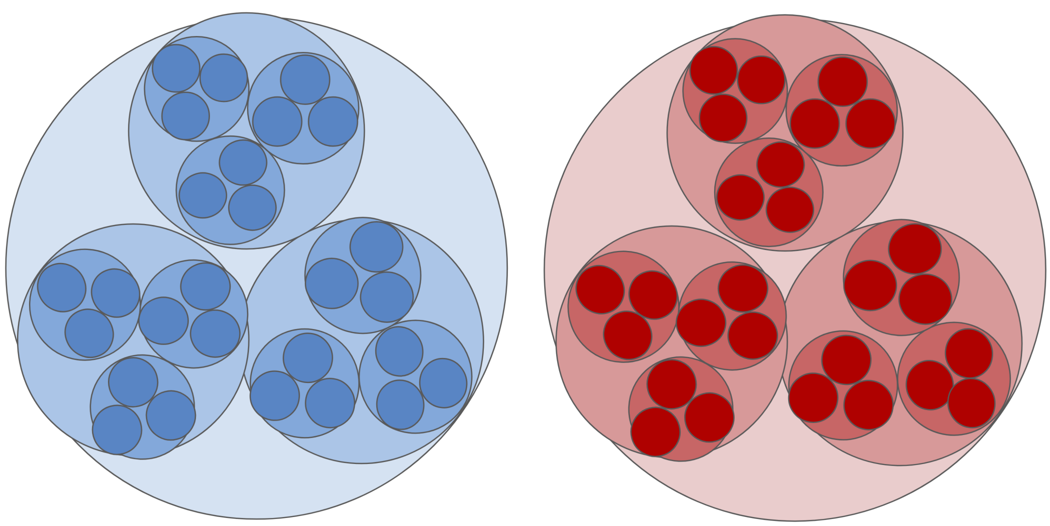
GM_bvh_2



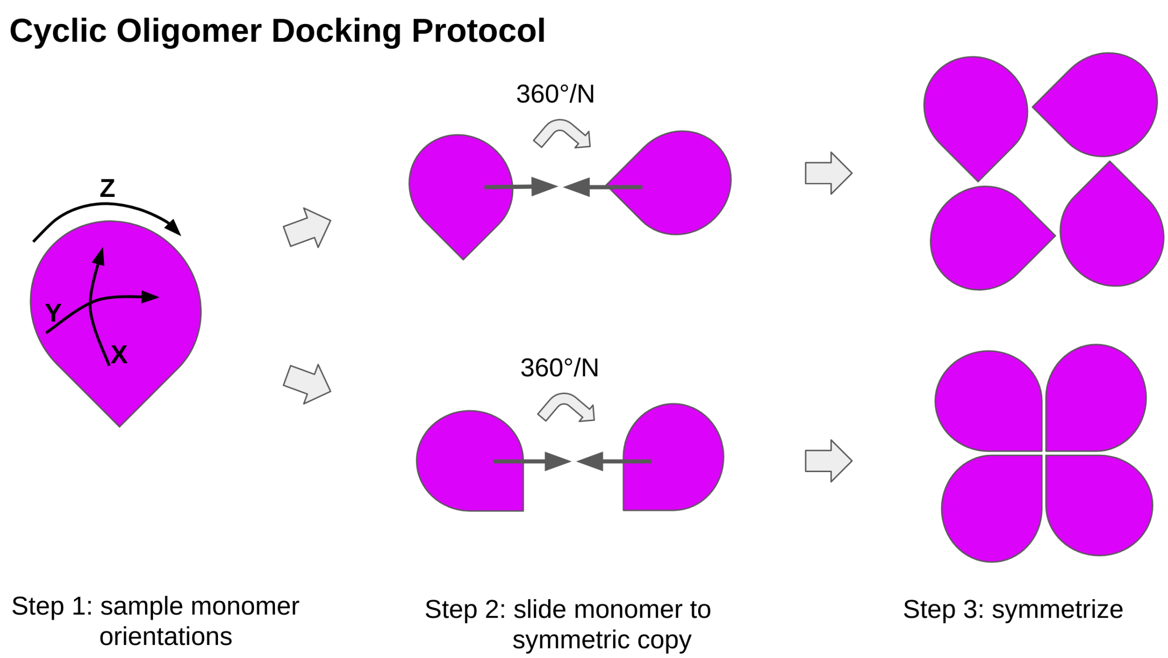
GM_bvh_3



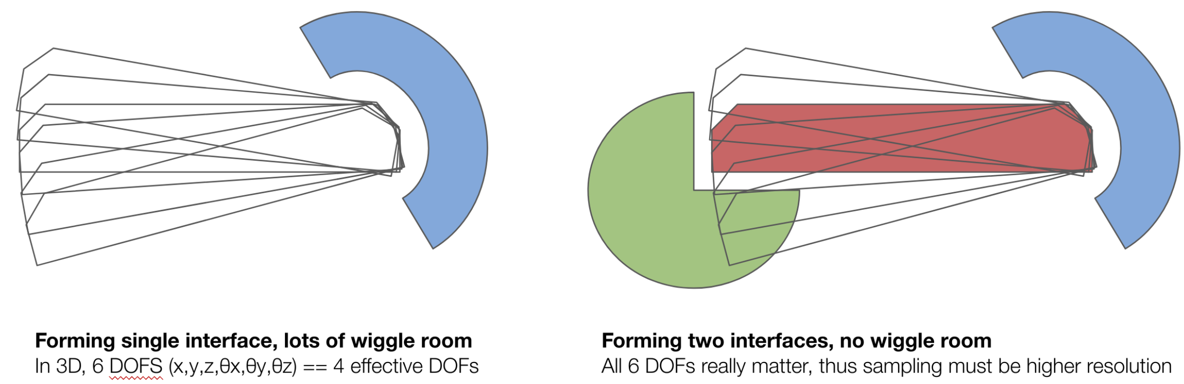
GM_bvh_4



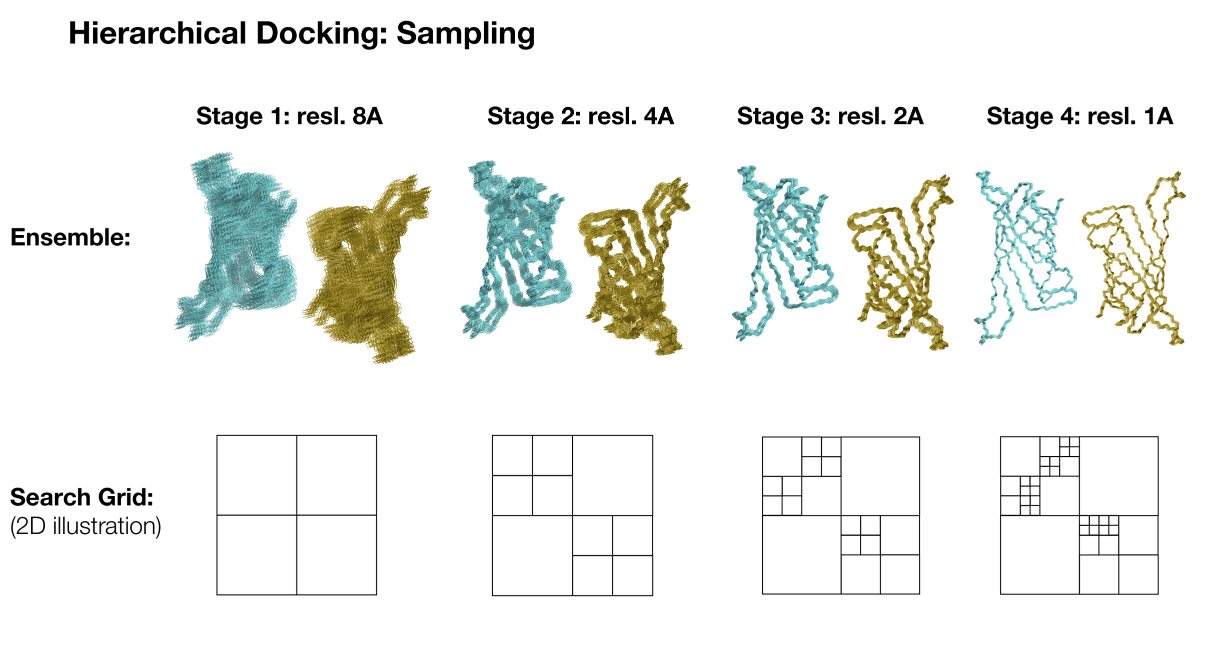
GM_dock_cyclic



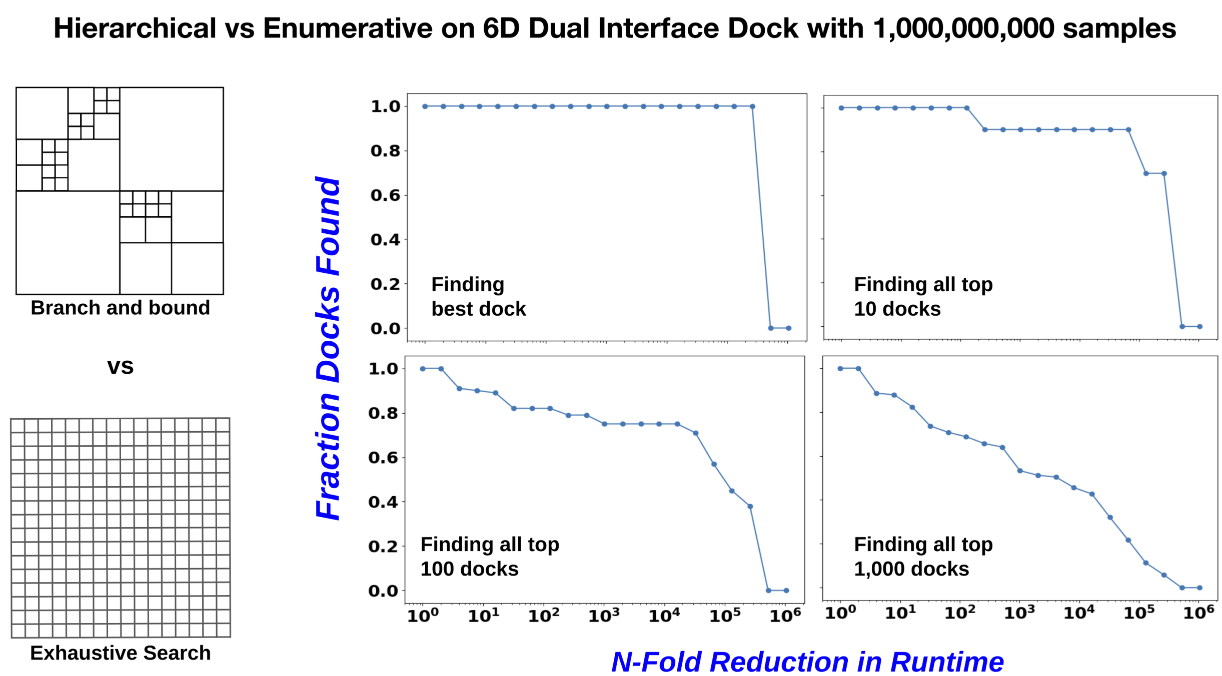
GM_forming_two_interfaces_requires_finer_search



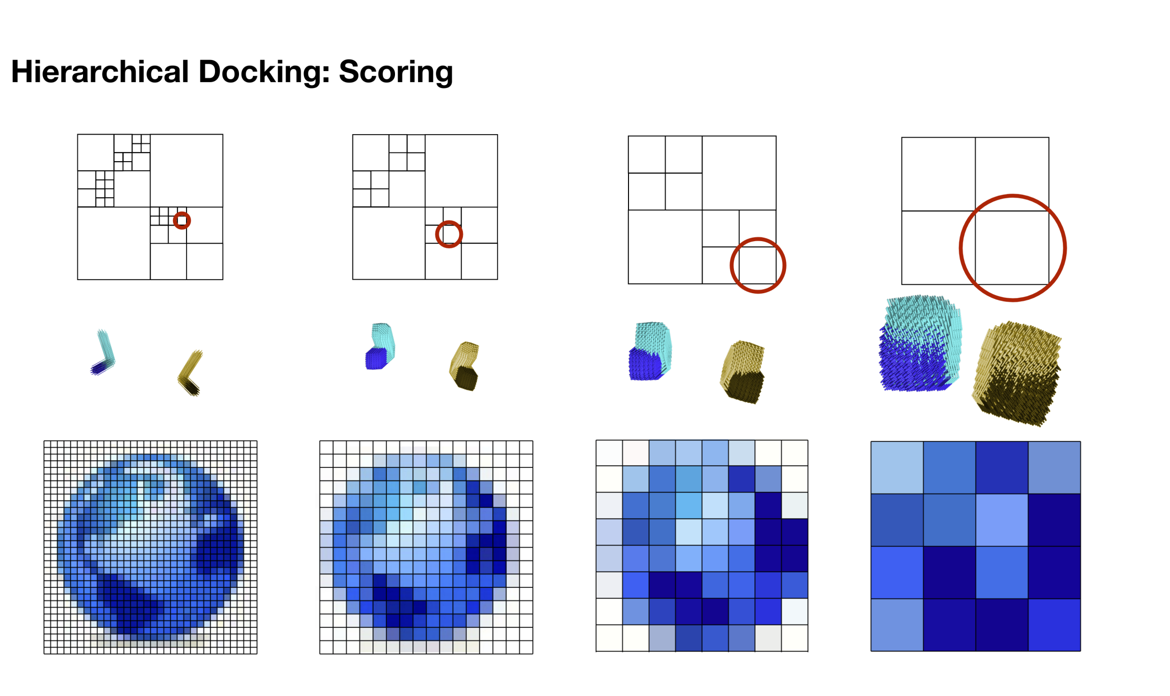
GM_hier_samp_ensemble



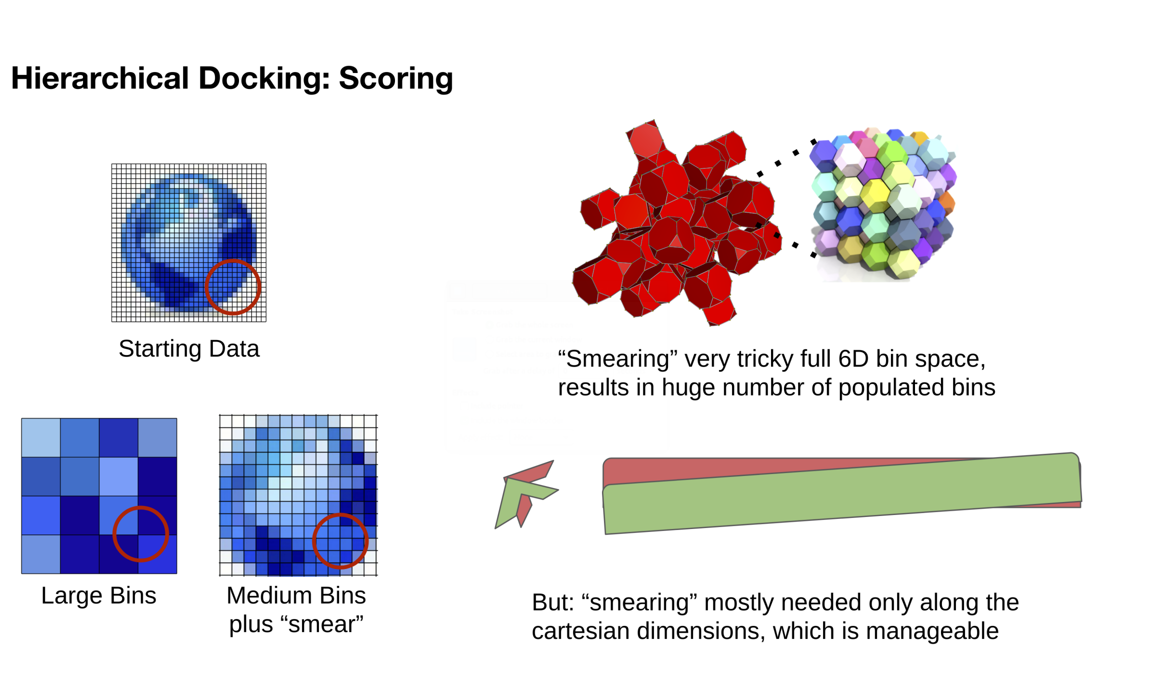
GM_hier_search_bench_plugdock



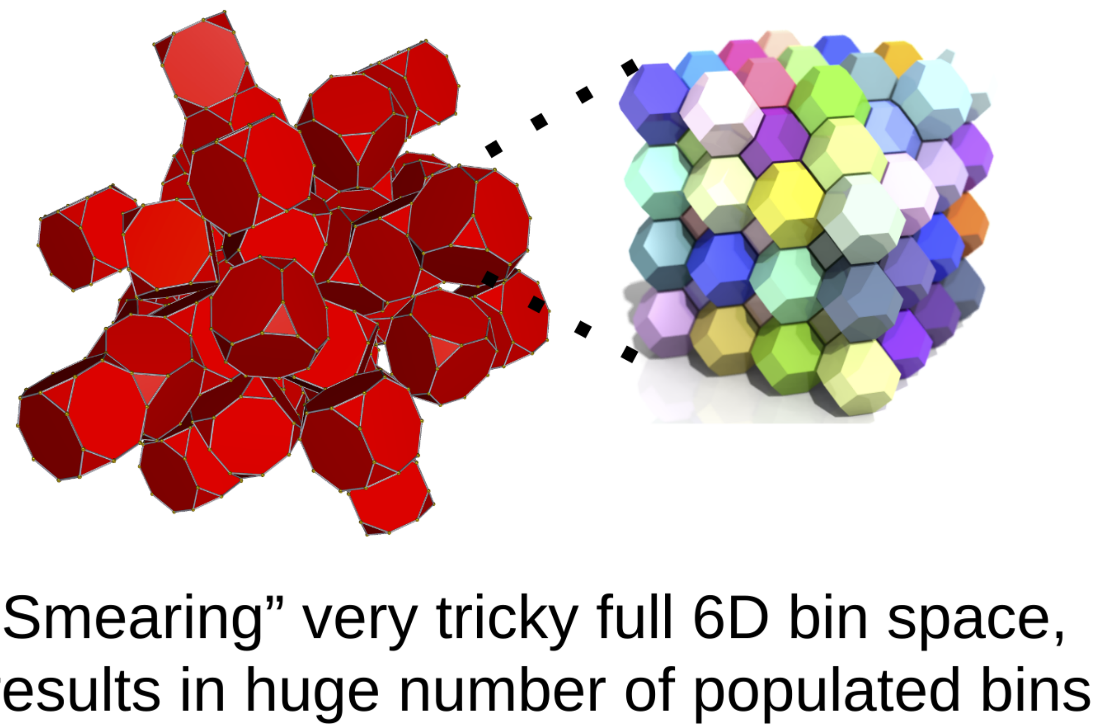
GM_hier_search_ensemble_score



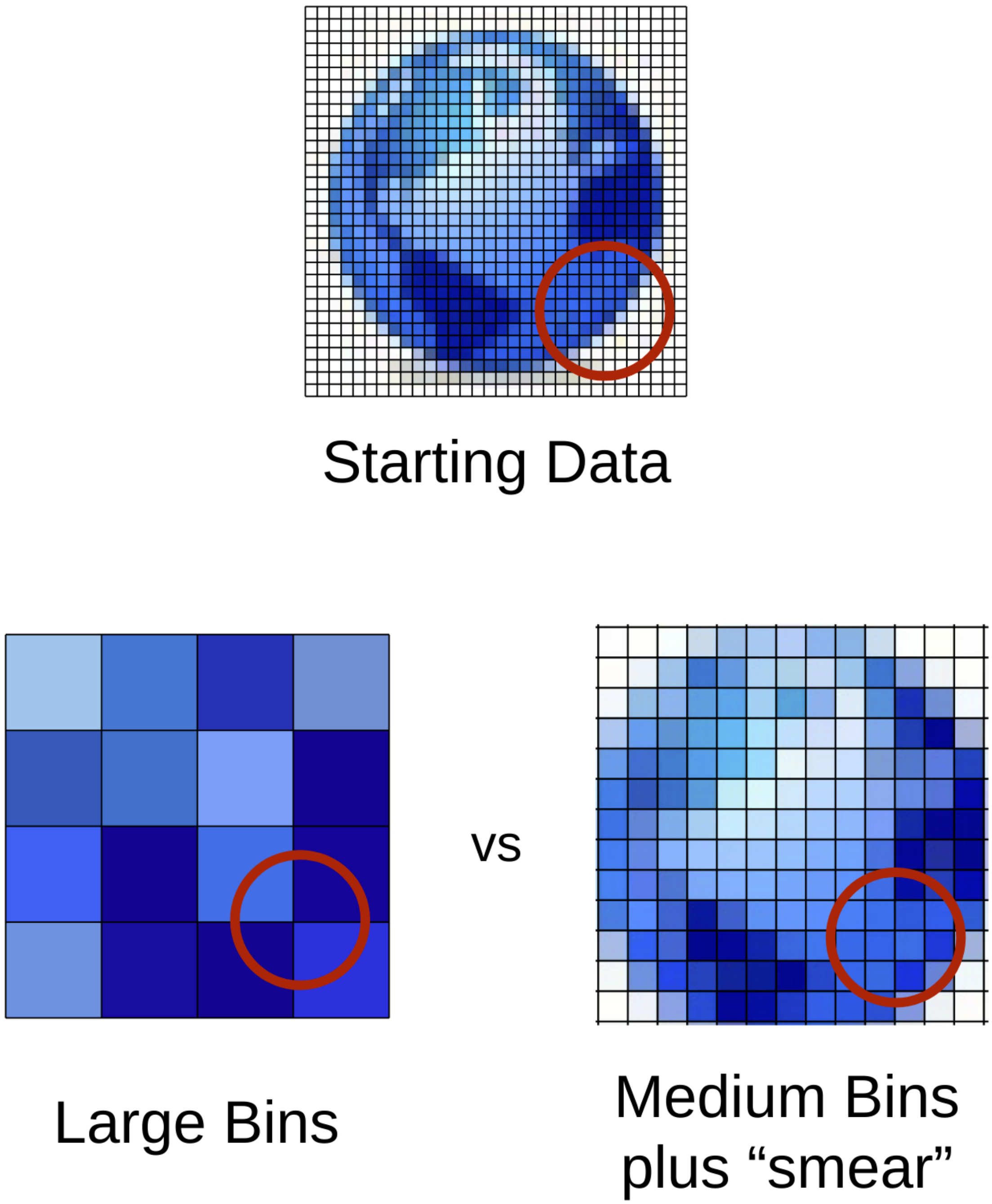
GM_hier_search_grids



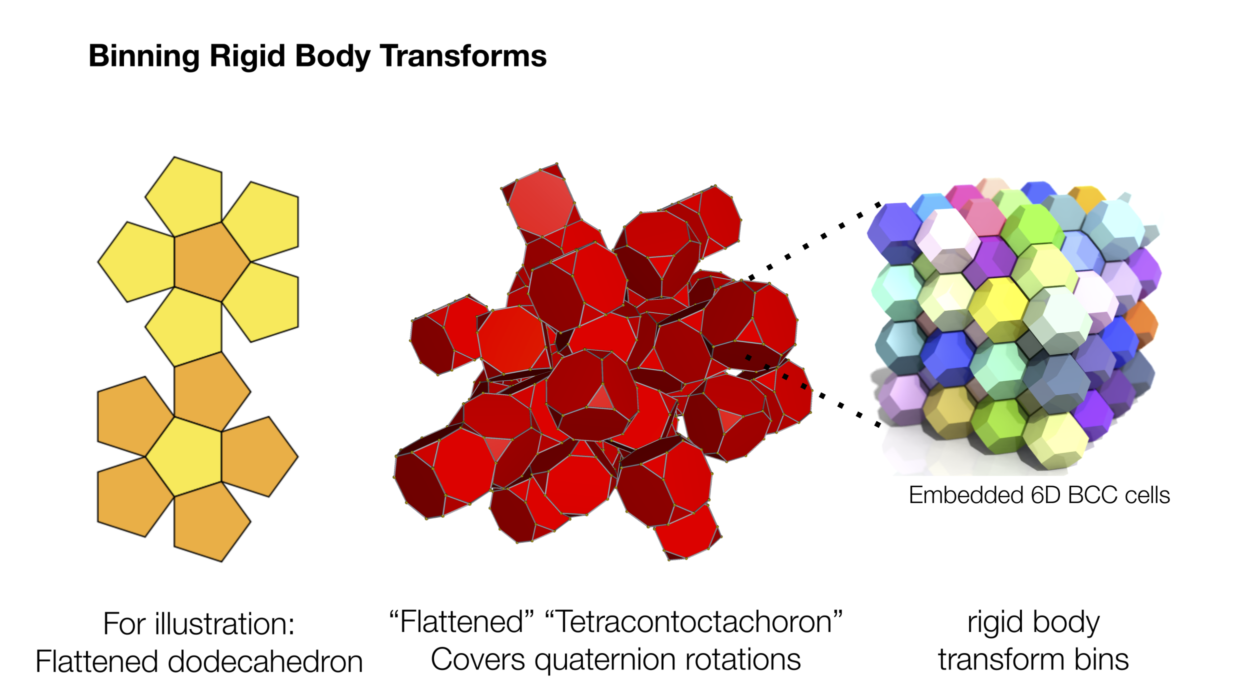
GM_hier_search_smear_6D



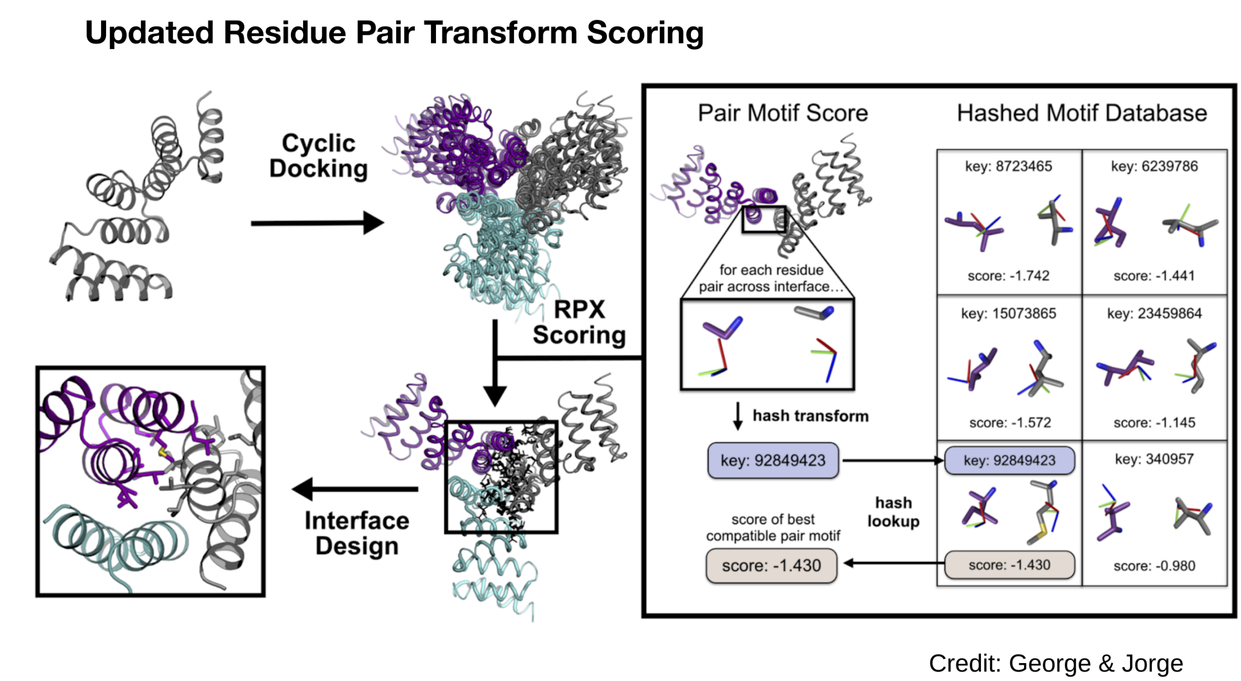
GM_hier_search_smear_illustration



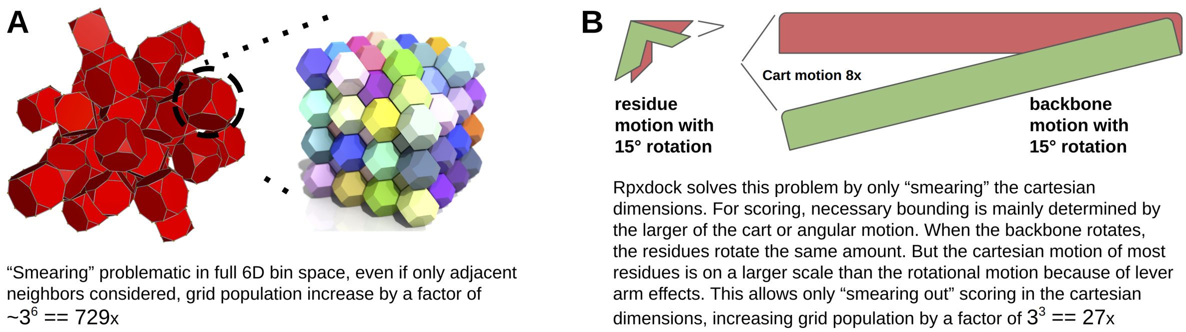
GM_rpx_bins



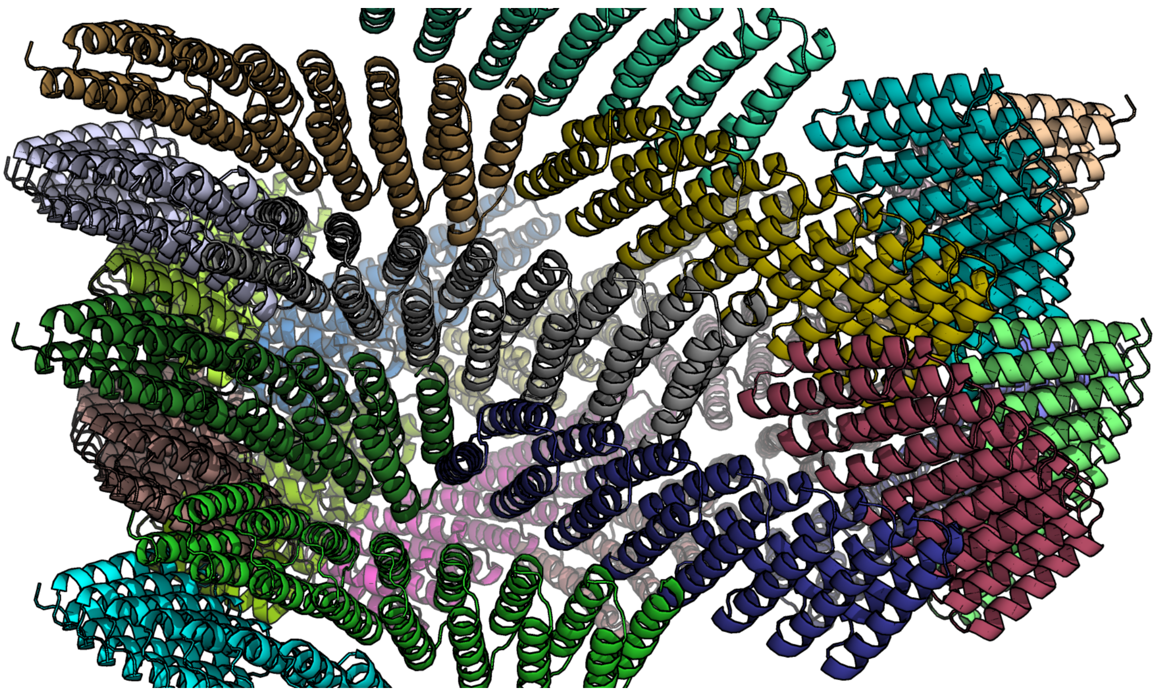
GM_rpx_jorge



GM_rpx_smear_cart_only

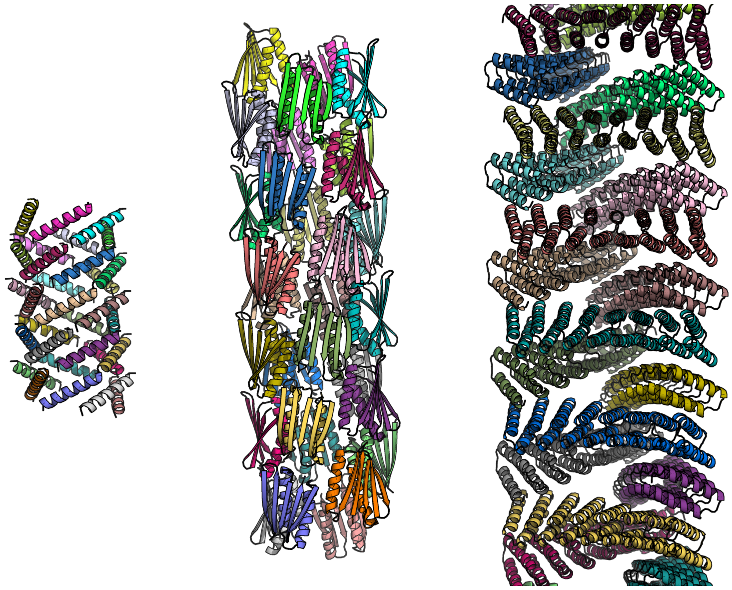


helix_dhr14_tube_brickwall



helix_from_repeat1

three_helix_outline



CHAPTER

SEVEN

RPXDOCK

7.1 rpxdock package

7.1.1 Subpackages

rpxdock.app package

Submodules

rpxdock.app.dock module

rpxdock.app.dock.**dock_cyclic**(*hscore*, *inputs*, *architecture*, ***kw*)

rpxdock.app.dock.**dock_multicomp**(*hscore*, ***kw*)

rpxdock.app.dock.**dock_onecomp**(*hscore*, ***kw*)

rpxdock.app.dock.**get_rpxdock_args**()

rpxdock.app.dock.**get_spec**(*arch*)

rpxdock.app.dock.**main**()

rpxdock.app.dump_results module

rpxdock.app.dump_results.**main**()

rpxdock.app.genrate_motif_scores module

rpxdock.app.genrate_motif_scores.**get_opts**()

rpxdock.app.genrate_motif_scores.**main**()

rpxdock.app.options module

rpxdock.app.options.**add_argument_unless_exists**(*parser*, **arg*, ***kw*)

rpxdock.app.options.**default_cli_parser**(*parent=None*, ***kw*)

rpxdock.app.options.**defaults**()

rpxdock.app.options.**get_cli_args**(*argv=None*, *parent=None*, ***kw*)

rpxdock.app.options.**make_argv_with_atfiles**(*argv=None*, ***kw*)

rpxdock.app.options.**parse_list_of_strtuple**(*s*)

rpxdock.app.options.**process_cart_bounds**(*cart_bounds*)

rpxdock.app.options.**process_cli_args**(*options*, ***kw*)

rpxdock.app.options.**set_loglevel**(*loglevel*)

rpxdock.app.options.**str2bool**(*v*)

rpxdock.app.rpx_score module

rpxdock.app.rpx_score.**get_opts**()

rpxdock.app.rpx_score.**main**()

rpxdock.app.rpx_score.**score_onebody**(*hscore*, ***kw*)

rpxdock.app.rpx_score.**score_twobody**(*hscore*, ***kw*)

Module contents

rpxdock.body package

Submodules

rpxdock.body.body module

class rpxdock.body.body.**Body**(*pdb_or_pose*, *sym='C1'*, *symaxis=[0, 0, 1]*, ***kw*)

Bases: object

clash_ok(**args*, ***kw*)

com()

contact_count(*other*, *maxdis*)

contact_pairs(*other*, *maxdis*, *buf=None*, *use_bb=False*, *atomno=False*)

copy()

copy_with_sym(*sym*, *symaxis=[0, 0, 1]*)

copy_xformed(*xform*)

distance_to(*other*)

dump_pdb(*fname*, ***kw*)

filter_pairs(*pairs*, *score_only_sspair*, *other=None*, *lbub=None*, *sanity_check=True*)

init_coords (*sym*, *symaxis*, *xform=array([[1.0, 0.0, 0.0, 0.0], [0.0, 1.0, 0.0, 0.0], [0.0, 0.0, 1.0, 0.0], [0.0, 0.0, 0.0, 1.0]])*)

intersect (*other*, *xself=None*, *xother=None*, *mindis=3.5*, ***kw*)

intersect_range (*other*, *xself=None*, *xother=None*, *mindis=3.5*, *max_trim=100*, *nasym1=None*, *debug=False*, ***kw*)

long_axis ()

long_axis_z_angle ()

move_by (*x*)

move_to (*x*)

move_to_center ()

positioned_cen (*asym=False*)

positioned_coord (*asym=False*)

positioned_coord_atomno (*i*)

positioned_orig_coords ()

radius_max ()

radius_xy_max ()

rg ()

rg_xy ()

rg_z ()

set_asym_body (*pose*, *sym*, ***kw*)

slide_to (*other*, *dirn*, *radius=1.75*)

str_pdb (***kw*)

strip_data ()

Module contents

rpxdock.bvh package

Submodules

rpxdock.bvh.bvh module

class `rpxdock.bvh.bvh.`**SphereBVH_double**

Bases: `pybind11_builtins.pybind11_object`

center (*self: rpxdock.bvh.bvh.SphereBVH_double*) → numpy.ndarray[float64[3, 1]]

centers (*self: rpxdock.bvh.bvh.SphereBVH_double*) → numpy.ndarray[float64]

com (*self: rpxdock.bvh.bvh.SphereBVH_double*) → numpy.ndarray[float64[4, 1]]

max_id (*self: rpxdock.bvh.bvh.SphereBVH_double*) → int

max_ub (*self: rpxdock.bvh.bvh.SphereBVH_double*) → int

min_lb (*self: rpxdock.bvh.bvh.SphereBVH_double*) → int

obj_id(*self: rpxdock.bvh.bvh.SphereBVH_double*) → numpy.ndarray[int32[m, 1]]

radius(*self: rpxdock.bvh.bvh.SphereBVH_double*) → float

vol_lb(*self: rpxdock.bvh.bvh.SphereBVH_double*) → numpy.ndarray[int32[m, 1]]

vol_ub(*self: rpxdock.bvh.bvh.SphereBVH_double*) → numpy.ndarray[int32[m, 1]]

rpxdock.bvh.bvh.**bvh_collect_pairs**(*arg0: rpxdock.bvh.bvh.SphereBVH_double, arg1: rpxdock.bvh.bvh.SphereBVH_double, arg2: numpy.ndarray[float64[4, 4]], arg3: numpy.ndarray[float64[4, 4]], arg4: float, arg5: numpy.ndarray[int32]*) → tuple

rpxdock.bvh.bvh.**bvh_collect_pairs_range_vec**(**args, **kwargs*)
Overloaded function.

1. bvh_collect_pairs_range_vec(bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float32], pos2: numpy.ndarray[float32], maxdist: float, lb1: numpy.ndarray[int32[m, 1]] = array([-2147483648], dtype=int32), ub1: numpy.ndarray[int32[m, 1]] = array([2147483647], dtype=int32), nasym1: int = -1, lb2: numpy.ndarray[int32[m, 1]] = array([-2147483648], dtype=int32), ub2: numpy.ndarray[int32[m, 1]] = array([2147483647], dtype=int32), nasym2: int = -1) -> tuple
2. bvh_collect_pairs_range_vec(bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float64], pos2: numpy.ndarray[float64], maxdist: float, lb1: numpy.ndarray[int32[m, 1]] = array([-2147483648], dtype=int32), ub1: numpy.ndarray[int32[m, 1]] = array([2147483647], dtype=int32), nasym1: int = -1, lb2: numpy.ndarray[int32[m, 1]] = array([-2147483648], dtype=int32), ub2: numpy.ndarray[int32[m, 1]] = array([2147483647], dtype=int32), nasym2: int = -1) -> tuple

rpxdock.bvh.bvh.**bvh_collect_pairs_vec**(**args, **kwargs*)
Overloaded function.

1. bvh_collect_pairs_vec(arg0: rpxdock.bvh.bvh.SphereBVH_double, arg1: rpxdock.bvh.bvh.SphereBVH_double, arg2: numpy.ndarray[float32], arg3: numpy.ndarray[float32], arg4: float) -> tuple
2. bvh_collect_pairs_vec(arg0: rpxdock.bvh.bvh.SphereBVH_double, arg1: rpxdock.bvh.bvh.SphereBVH_double, arg2: numpy.ndarray[float64], arg3: numpy.ndarray[float64], arg4: float) -> tuple

rpxdock.bvh.bvh.**bvh_count_pairs**(*arg0: rpxdock.bvh.bvh.SphereBVH_double, arg1: rpxdock.bvh.bvh.SphereBVH_double, arg2: numpy.ndarray[float64[4, 4]], arg3: numpy.ndarray[float64[4, 4]], arg4: float*) → int

rpxdock.bvh.bvh.**bvh_count_pairs_vec**(*arg0: rpxdock.bvh.bvh.SphereBVH_double, arg1: rpxdock.bvh.bvh.SphereBVH_double, arg2: numpy.ndarray[float64], arg3: numpy.ndarray[float64], arg4: float*) → numpy.ndarray[int32[m, 1]]

rpxdock.bvh.bvh.**bvh_isect**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float64[4, 4]], pos2: numpy.ndarray[float64[4, 4]], mindist: float*) → bool
interstion test

rpxdock.bvh.bvh.**bvh_isect_fixed**(*arg0: rpxdock.bvh.bvh.SphereBVH_double, arg1: rpxdock.bvh.bvh.SphereBVH_double, arg2: float*) → bool `rpxdock.bvh.bvh.`**`bvh_isect_fixed_range_vec`**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float64], pos2: numpy.ndarray[float64], mindist: float, lb1: numpy.ndarray[int32[m, 1]] = array([0], dtype=int32), ub1: numpy.ndarray[int32[m, 1]] = array([99999999], dtype=int32), lb2: numpy.ndarray[int32[m, 1]] = array([0], dtype=int32), ub2: numpy.ndarray[int32[m, 1]] = array([99999999], dtype=int32)*) → tuple

intersction test with input range

`rpxdock.bvh.bvh.`**`bvh_isect_vec`**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float64], pos2: numpy.ndarray[float64], mindist: float*) → numpy.ndarray[bool[m, 1]]

intersction test

`rpxdock.bvh.bvh.`**`bvh_min_dist`**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float64[4, 4]], pos2: numpy.ndarray[float64[4, 4]]*) → tuple

min pair distance

`rpxdock.bvh.bvh.`**`bvh_min_dist_fixed`**(*arg0: rpxdock.bvh.bvh.SphereBVH_double, arg1: rpxdock.bvh.bvh.SphereBVH_double*) → tuple

`rpxdock.bvh.bvh.`**`bvh_min_dist_one`**(*arg0: rpxdock.bvh.bvh.SphereBVH_double, arg1: numpy.ndarray[float64[3, 1]]*) → tuple

`rpxdock.bvh.bvh.`**`bvh_min_dist_vec`**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float64], pos2: numpy.ndarray[float64]*) → tuple

min pair distance

`rpxdock.bvh.bvh.`**`bvh_print`**(*arg0: rpxdock.bvh.bvh.SphereBVH_double*) → int

`rpxdock.bvh.bvh.`**`bvh_slide`**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float64[4, 4]], pos2: numpy.ndarray[float64[4, 4]], rad: float, dirn: numpy.ndarray[float64[3, 1]]*) → float

slide into contact

`rpxdock.bvh.bvh.`**`bvh_slide_vec`**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float64], pos2: numpy.ndarray[float64], rad: float, dirn: numpy.ndarray[float64[3, 1]]*) → numpy.ndarray[float64[m, 1]]

slide into contact

`rpxdock.bvh.bvh.`**`isect_range`**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double, bvh2: rpxdock.bvh.bvh.SphereBVH_double, pos1: numpy.ndarray[float64], pos2: numpy.ndarray[float64], mindist: float, maxtrim: int = - 1, maxtrim_lb: int = - 1, maxtrim_ub: int = - 1, nasym1: int = - 1*) → tuple

intersction test

rpxdock.bvh.bvh.**isect_range_single**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double*, *bvh2: rpxdock.bvh.bvh.SphereBVH_double*, *pos1: numpy.ndarray[float64[4, 4]]*, *pos2: numpy.ndarray[float64[4, 4]]*, *mindist: float*, *maxtrim: int = - 1*, *maxtrim_lb: int = - 1*, *maxtrim_ub: int = - 1*, *nasym1: int = - 1*) → tuple

intersction test

rpxdock.bvh.bvh.**naive_collect_pairs**(*arg0: rpxdock.bvh.bvh.SphereBVH_double*, *arg1: rpxdock.bvh.bvh.SphereBVH_double*, *arg2: numpy.ndarray[float64[4, 4]]*, *arg3: numpy.ndarray[float64[4, 4]]*, *arg4: float*, *arg5: numpy.ndarray[int32]*) → int

rpxdock.bvh.bvh.**naive_isect**(*arg0: rpxdock.bvh.bvh.SphereBVH_double*, *arg1: rpxdock.bvh.bvh.SphereBVH_double*, *arg2: numpy.ndarray[float64[4, 4]]*, *arg3: numpy.ndarray[float64[4, 4]]*, *arg4: float*) → bool

rpxdock.bvh.bvh.**naive_isect_fixed**(*arg0: rpxdock.bvh.bvh.SphereBVH_double*, *arg1: rpxdock.bvh.bvh.SphereBVH_double*, *arg2: float*) → bool

rpxdock.bvh.bvh.**naive_isect_range**(*bvh1: rpxdock.bvh.bvh.SphereBVH_double*, *bvh2: rpxdock.bvh.bvh.SphereBVH_double*, *pos1: numpy.ndarray[float64[4, 4]]*, *pos2: numpy.ndarray[float64[4, 4]]*, *mindist: float*) → tuple

intersction test

rpxdock.bvh.bvh.**naive_min_dist**(*arg0: rpxdock.bvh.bvh.SphereBVH_double*, *arg1: rpxdock.bvh.bvh.SphereBVH_double*, *arg2: numpy.ndarray[float64[4, 4]]*, *arg3: numpy.ndarray[float64[4, 4]]*) → float

rpxdock.bvh.bvh.**naive_min_dist_fixed**(*arg0: rpxdock.bvh.bvh.SphereBVH_double*, *arg1: rpxdock.bvh.bvh.SphereBVH_double*) → float

rpxdock.bvh.bvh_nd module

class rpxdock.bvh.bvh_nd.**SphereBVH4D**

Bases: `pybind11_builtins.pybind11_object`

center(*self: rpxdock.bvh.bvh_nd.SphereBVH4D*) → numpy.ndarray[float64[4, 1]]

centers(*self: rpxdock.bvh.bvh_nd.SphereBVH4D*) → numpy.ndarray[float64[m, 4]]

com(*self: rpxdock.bvh.bvh_nd.SphereBVH4D*) → numpy.ndarray[float64[4, 1]]

radius(*self: rpxdock.bvh.bvh_nd.SphereBVH4D*) → float

class rpxdock.bvh.bvh_nd.**SphereBVH7D**

Bases: `pybind11_builtins.pybind11_object`

center(*self: rpxdock.bvh.bvh_nd.SphereBVH7D*) → numpy.ndarray[float64[7, 1]]

centers(*self: rpxdock.bvh.bvh_nd.SphereBVH7D*) → numpy.ndarray[float64[m, 7]]

com(*self: rpxdock.bvh.bvh_nd.SphereBVH7D*) → numpy.ndarray[float64[7, 1]]

radius(*self: rpxdock.bvh.bvh_nd.SphereBVH7D*) → float

rpxdock.bvh.bvh_nd.**bvh_bvh_isect7d**(*arg0: rpxdock.bvh.bvh_nd.SphereBVH7D*, *arg1: rpxdock.bvh.bvh_nd.SphereBVH7D*, *arg2: float*) → bool rpxdock.bvh.bvh_nd.**bvh_bvh_isect7d_naive**(*arg0:* rpxdock.bvh.bvh_nd.SphereBVH7D, *arg1:* rpxdock.bvh.bvh_nd.SphereBVH7D, *arg2: float*) → bool

rpxdock.bvh.bvh_nd.**bvh_isect7d**(*arg0:* rpxdock.bvh.bvh_nd.SphereBVH7D, *arg1: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*, *arg2: float*) → numpy.ndarray[int32[m, 1]]

rpxdock.bvh.bvh_nd.**bvh_isect7d_naive**(*arg0:* rpxdock.bvh.bvh_nd.SphereBVH7D, *arg1: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*, *arg2: float*) → numpy.ndarray[int32[m, 1]]

rpxdock.bvh.bvh_nd.**bvh_mindist4d**(*arg0:* rpxdock.bvh.bvh_nd.SphereBVH4D, *arg1: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → tuple

rpxdock.bvh.bvh_nd.**bvh_mindist4d_naive**(*arg0:* rpxdock.bvh.bvh_nd.SphereBVH4D, *arg1: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → tuple

rpxdock.bvh.bvh_nd.**bvh_mindist7d**(*arg0:* rpxdock.bvh.bvh_nd.SphereBVH7D, *arg1: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → tuple

rpxdock.bvh.bvh_nd.**bvh_mindist7d_naive**(*arg0:* rpxdock.bvh.bvh_nd.SphereBVH7D, *arg1: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → tuple

rpxdock.bvh.bvh_nd.**create_bvh4d**(*arg0: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → *rpxdock.bvh.bvh_nd.SphereBVH4D*

rpxdock.bvh.bvh_nd.**create_bvh7d**(*arg0: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → *rpxdock.bvh.bvh_nd.SphereBVH7D*

rpxdock.bvh.bvh_nd.**create_bvh_quat**(*arg0: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → *rpxdock.bvh.bvh_nd.SphereBVH4D*

rpxdock.bvh.bvh_nd.**create_bvh_xform**(*arg0: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → *rpxdock.bvh.bvh_nd.SphereBVH7D*

rpxdock.bvh.bvh_test module

rpxdock.bvh.bvh_test.**TEST_bvh_test_isect**() → bool

rpxdock.bvh.bvh_test.**TEST_bvh_test_min**() → bool

Module contents

rpxdock.cluster package

Submodules

rpxdock.cluster.cookie_cutter module

rpxdock.cluster.cookie_cutter.**cookie_cutter**(**args*, ***kwargs*)
Overloaded function.

1. cookie_cutter(arg0: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous], arg1: float) -> numpy.ndarray[int32[1, n]]
2. cookie_cutter(arg0: numpy.ndarray[float32[m, n], flags.writeable, flags.c_contiguous], arg1: float) -> numpy.ndarray[int32[1, n]]

rpxdock.cluster.prune module

`rpxdock.cluster.prune.`**`extract_2comp_dofs`**(*spec*, *pos*)

`rpxdock.cluster.prune.`**`prune_results_2comp`**(*spec*, *body1*, *body2*, *score*, *pos*, *mindis=5*)

Module contents

rpxdock.data package

Submodules

rpxdock.data.data module

`rpxdock.data.data.`**`get_body`**(*name*)

`rpxdock.data.data.`**`get_test_data`**(*name*)

`rpxdock.data.data.`**`small_hscore`**()

`rpxdock.data.data.`**`small_respairdat`**()

`rpxdock.data.data.`**`small_respairscore`**()

Module contents

rpxdock.filter package

Submodules

rpxdock.filter.redundancy module

`rpxdock.filter.redundancy.`**`filter_redundancy`**(*xforms*, *body*, *scores=None*, *categories=None*, *every_nth=10*, ***kw*)

Module contents

rpxdock.geom package

Submodules

rpxdock.geom.bcc module

class `rpxdock.geom.bcc.`**`BCC3`**

Bases: `pybind11_builtins.pybind11_object`

keys(*self: rpxdock.geom.bcc.BCC3, arg0: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → numpy.ndarray[uint64[m, 1]]

property lower

neighbor_radius_square_cut(*self: rpxdock.geom.bcc.BCC3, radius: int, extrahalf: bool = False*) → int

neighbor_sphere_radius_square_cut(*self: rpxdock.geom.bcc.BCC3, radius: int, extrahalf: bool = False*) → int

neighbors_3(*self: rpxdock.geom.bcc.BCC3, index: int, radius: int = 1, extrahalf: bool = False, sphere: bool = True*) → numpy.ndarray[uint64[m, 1]]
get indices of neighboring cells

neighbors_3_dist(*self: rpxdock.geom.bcc.BCC3, index: int, radius: int = 1, extrahalf: bool = False, sphere: bool = True*) → tuple
get indices of neighboring cells

neighbors_6_3_dist(*self: rpxdock::geom::BCC<6, double, unsigned long>, index: int, radius: int = 1, extrahalf: bool = False, oddlast3: bool = True, sphere: bool = True*) → tuple
get indices of neighboring cells, last3 dims only +-1

property nside

property upper

vals(*self: rpxdock.geom.bcc.BCC3, arg0: numpy.ndarray[uint64[m, 1], flags.writeable]*) → numpy.ndarray[float64[m, n]]

property width

class rpxdock.geom.bcc.**BCC3_float**
Bases: pybind11_builtins.pybind11_object

keys(*self: rpxdock.geom.bcc.BCC3_float, arg0: numpy.ndarray[float32[m, n], flags.writeable, flags.c_contiguous]*) → numpy.ndarray[uint64[m, 1]]

property lower

neighbor_radius_square_cut(*self: rpxdock.geom.bcc.BCC3_float, radius: int, extrahalf: bool = False*) → int

neighbor_sphere_radius_square_cut(*self: rpxdock.geom.bcc.BCC3_float, radius: int, extrahalf: bool = False*) → int

neighbors_3(*self: rpxdock.geom.bcc.BCC3_float, index: int, radius: int = 1, extrahalf: bool = False, sphere: bool = True*) → numpy.ndarray[uint64[m, 1]]
get indices of neighboring cells

neighbors_3_dist(*self: rpxdock.geom.bcc.BCC3_float, index: int, radius: int = 1, extrahalf: bool = False, sphere: bool = True*) → tuple
get indices of neighboring cells

neighbors_6_3_dist(*self: rpxdock::geom::BCC<6, float, unsigned long>, index: int, radius: int = 1, extrahalf: bool = False, oddlast3: bool = True, sphere: bool = True*) → tuple
get indices of neighboring cells, last3 dims only +-1

property nside

property upper

vals(*self: rpxdock.geom.bcc.BCC3_float, arg0: numpy.ndarray[uint64[m, 1], flags.writeable]*) → numpy.ndarray[float32[m, n]]

property width

class rpxdock.geom.bcc.**BCC6**
Bases: pybind11_builtins.pybind11_object

keys (*self: rpxdock.geom.bcc.BCC6*, *arg0: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → numpy.ndarray[uint64[m, 1]]

property lower

neighbor_radius_square_cut (*self: rpxdock.geom.bcc.BCC6*, *radius: int*, *extrahalf: bool = False*) → int

neighbor_sphere_radius_square_cut (*self: rpxdock.geom.bcc.BCC6*, *radius: int*, *extrahalf: bool = False*) → int

neighbors_3_dist (*self: rpxdock.geom.bcc.BCC3*, *index: int*, *radius: int = 1*, *extrahalf: bool = False*, *sphere: bool = True*) → tuple
get indices of neighboring cells

neighbors_6_3 (*self: rpxdock.geom.bcc.BCC6*, *index: int*, *radius: int = 1*, *extrahalf: bool = False*, *oddlast3: bool = True*, *sphere: bool = True*) → numpy.ndarray[uint64[m, 1]]
get indices of neighboring cells, last3 dims only +-1

neighbors_6_3_dist (*self: rpxdock.geom.bcc.BCC6*, *index: int*, *radius: int = 1*, *extrahalf: bool = False*, *oddlast3: bool = True*, *sphere: bool = True*) → tuple
get indices of neighboring cells, last3 dims only +-1

property nside

property upper

vals (*self: rpxdock.geom.bcc.BCC6*, *arg0: numpy.ndarray[uint64[m, 1], flags.writeable]*) → numpy.ndarray[float64[m, n]]

property width

class rpxdock.geom.bcc.**BCC6_float**
Bases: pybind11_builtins.pybind11_object

keys (*self: rpxdock.geom.bcc.BCC6_float*, *arg0: numpy.ndarray[float32[m, n], flags.writeable, flags.c_contiguous]*) → numpy.ndarray[uint64[m, 1]]

property lower

neighbor_radius_square_cut (*self: rpxdock.geom.bcc.BCC6_float*, *radius: int*, *extrahalf: bool = False*) → int

neighbor_sphere_radius_square_cut (*self: rpxdock.geom.bcc.BCC6_float*, *radius: int*, *extrahalf: bool = False*) → int

neighbors_3_dist (*self: rpxdock.geom.bcc.BCC3_float*, *index: int*, *radius: int = 1*, *extrahalf: bool = False*, *sphere: bool = True*) → tuple
get indices of neighboring cells

neighbors_6_3 (*self: rpxdock.geom.bcc.BCC6_float*, *index: int*, *radius: int = 1*, *extrahalf: bool = False*, *oddlast3: bool = True*, *sphere: bool = True*) → numpy.ndarray[uint64[m, 1]]
get indices of neighboring cells, last3 dims only +-1

neighbors_6_3_dist (*self: rpxdock.geom.bcc.BCC6_float*, *index: int*, *radius: int = 1*, *extrahalf: bool = False*, *oddlast3: bool = True*, *sphere: bool = True*) → tuple
get indices of neighboring cells, last3 dims only +-1

property nside

property upper

vals (*self: rpxdock.geom.bcc.BCC6_float*, *arg0: numpy.ndarray[uint64[m, 1], flags.writeable]*) → numpy.ndarray[float32[m, n]]

property width

rpxdock.geom.miniball module

`rpxdock.geom.miniball.`**miniball**(*arg0: numpy.ndarray[float64[m, n], flags.writeable, flags.c_contiguous]*) → numpy.ndarray[float64[m, 1]]

`rpxdock.geom.miniball.`**miniball_test**(*arg0: int*, *arg1: int*, *arg2: bool*) → bool

rpxdock.geom.porosity module

`rpxdock.geom.porosity.`**sphere_porosity**(*ca*, *sym*, ***kw*)

rpxdock.geom.primitive_test module

`rpxdock.geom.primitive_test.`**TEST_geom_primitive_sphere**() → bool

`rpxdock.geom.primitive_test.`**TEST_geom_primitive_welzl_bounding_sphere**() → bool

rpxdock.geom.sym module

`rpxdock.geom.sym.`**symframes**(*sym*, *pos=None*, *axis=[0, 0, 1]*, ***kw*)

rpxdock.geom.xform_dist module

`rpxdock.geom.xform_dist.`**xform_dist2_split**(**args*, ***kwargs*)

Overloaded function.

1. xform_dist2_split(arg0: numpy.ndarray[float32], arg1: numpy.ndarray[float32], arg2: float) -> tuple
2. xform_dist2_split(arg0: numpy.ndarray[float64], arg1: numpy.ndarray[float64], arg2: float) -> tuple

Module contents

`rpxdock.geom.`**xform_dist2**(**args*)

rpxdock.io package

Submodules

rpxdock.io.io module

`rpxdock.io.io.`**aname_to_elem**(*aname*)

return based on first occurance of element letter

`rpxdock.io.io.`**dump_pdb_from_points**(*fname*, *pts*)

`rpxdock.io.io.`**pdb_format_atom**(*ia=0*, *an='ATOM'*, *idx=' '*, *rn='RES'*, *c='A'*, *ir=0*, *insert=' '*, *x=0*, *y=0*, *z=0*, *occ=1*, *b=1*, *elem=' '*, *xyz=None*)

rpxdock.io.io_body module

rpxdock.io.io_body.**dump_pdb_from_bodies**(*fname*, **args*, ***kw*)

rpxdock.io.io_body.**make_pdb_from_bodies**(*bodies*, *symframes=None*, *start=(0, 0)*, *use_body_sym=None*, *keep=<function <lambda>>*, *no_duplicate_chains=False*, *no_duplicate_reschain_pairs=True*, *include_cen=True*, *only_atoms=None*, *chain_letters=None*, *resbounds=[]*, *bfactor=None*, *occupancy=None*, *use_orig_coords=False*, *warn_on_chain_overflow=True*, ***kw*)

Module contents

rpxdock.motif package

Submodules

rpxdock.motif.frames module

rpxdock.motif.frames.**add_rots_to_respairdat**(*rp*, *rotspace*, ***kw*)

rpxdock.motif.frames.**add_xbin_to_respairdat**(*rp*, *xbin*, ***kw*)

rpxdock.motif.frames.**bb_stubs**(*n*, *ca=None*, *c=None*, *dtype='f4'*)

rpxdock.motif.frames.**get_pair_keys**(*rp*, *xbin*, *min_pair_score*, *min_ssep*, *use_ss_key*, ***kw*)

rpxdock.motif.frames.**make_respairdat_subsets**(*rp*)

rpxdock.motif.frames.**remove_redundant_pdbs**(*pdbs*, *sequence_identity=30*)

rpxdock.motif.frames.**ss_to_ssid**(*ss*)

rpxdock.motif.frames.**stub_from_points**(*cen*, *pa=None*, *pb=None*, *dtype='f4'*)

rpxdock.motif.pairdat module

class rpxdock.motif.pairdat.**ResPairData**(*data*, *sanity_check=None*)

Bases: object

only_whats_needed(*task*)

sanity_check()

split_by_pdb(*frac*, *random=True*, ***kw*)

subset_by_aa(*aas*, *sanity_check=False*, *return_keepers=False*)

subset_by_pair(*keepers*, *sanity_check=False*, *update_p_res=True*)

subset_by_pdb(*keep*, *sanity_check=False*, *update_p_res=True*, ***kw*)

keep subset of data in same order as original

subset_by_res(*keepers*, *sanity_check=False*)

subset_by_ss(*ss*, *sanity_check=False*, *return_keepers=False*)

rpxdock.motif.pairscore module

class rpxdock.motif.pairscore.**ResPairScore**(*xbin*, *keys*, *score_map*, *range_map*, *res1*, *res2*, *rotspace*, *rp*)

Bases: object

add_hier_score(*resl*, *scoremap*)

bin_get_all_data(*keys*)

bin_respairs(*key*)

bin_score(*keys*)

hier_score(*resl*)

class rpxdock.motif.pairscore.**Xmap**(*xbin*, *phmap*, *attrs={}*, *rehash_bincens=False*)

Bases: object

has(*x_or_k*)

key_of(*x_or_k*)

keys(*n=- 1*)

xforms(*n=- 1*)

rpxdock.motif.pairscore.**create_res_pair_score**(*rp*, *xbin*, ***kw*)

rpxdock.motif.pairscore.**create_res_pair_score_map**(*rp*, *xbin*, *min_bin_score*, ***kw*)

rpxdock.motif.rpxgen module

rpxdock.motif.rpxgen.**create_xbin_even_nside**(*cart_resl*, *ori_resl*, *max_cart*)

rpxdock.motif.rpxgen.**make_and_dump_hier_score_tables**(*pairdat*, ***kw*)

rpxdock.motif.rpxgen.**make_hscore_single**(*pairdat*, *ihier*, *xbin_base*, *cart_extent*, *ori_extent*, *sstag*, ***kw*)

Module contents

rpxdock.phmap package

Submodules

rpxdock.phmap.phmap module

class rpxdock.phmap.phmap.**PHMap_u4f4**

Bases: pybind11_builtins.pybind11_object

property default

has(*self: rpxdock.phmap.phmap.PHMap_u4f4*, *arg0: numpy.ndarray[uint32[m, 1], flags.writeable]*) → numpy.ndarray[bool[m, 1]]

items(*self: rpxdock.phmap.phmap.PHMap_u4f4*) → iterator

items_array(*self: rpxdock.phmap.phmap.PHMap_u4f4*, *num: int = - 1*) → tuple

keys(*self: rpxdock.phmap.phmap.PHMap_u4f4*, *num: int = - 1*) → numpy.ndarray[uint32[m, 1]]

class rpxdock.phmap.phmap.**PHMap_u8f4**
Bases: pybind11_builtins.pybind11_object

property default

has (*self: rpxdock.phmap.phmap.PHMap_u8f4*, *arg0: numpy.ndarray[uint64[m, 1], flags.writeable]*) → numpy.ndarray[bool[m, 1]]

items (*self: rpxdock.phmap.phmap.PHMap_u8f4*) → iterator

items_array (*self: rpxdock.phmap.phmap.PHMap_u8f4*, *num: int = - 1*) → tuple

keys (*self: rpxdock.phmap.phmap.PHMap_u8f4*, *num: int = - 1*) → numpy.ndarray[uint64[m, 1]]

class rpxdock.phmap.phmap.**PHMap_u8f8**
Bases: pybind11_builtins.pybind11_object

property default

has (*self: rpxdock.phmap.phmap.PHMap_u8f8*, *arg0: numpy.ndarray[uint64[m, 1], flags.writeable]*) → numpy.ndarray[bool[m, 1]]

items (*self: rpxdock.phmap.phmap.PHMap_u8f8*) → iterator

items_array (*self: rpxdock.phmap.phmap.PHMap_u8f8*, *num: int = - 1*) → tuple

keys (*self: rpxdock.phmap.phmap.PHMap_u8f8*, *num: int = - 1*) → numpy.ndarray[uint64[m, 1]]

class rpxdock.phmap.phmap.**PHMap_u8u8**
Bases: pybind11_builtins.pybind11_object

property default

has (*self: rpxdock.phmap.phmap.PHMap_u8u8*, *arg0: numpy.ndarray[uint64[m, 1], flags.writeable]*) → numpy.ndarray[bool[m, 1]]

items (*self: rpxdock.phmap.phmap.PHMap_u8u8*) → iterator

items_array (*self: rpxdock.phmap.phmap.PHMap_u8u8*, *num: int = - 1*) → tuple

keys (*self: rpxdock.phmap.phmap.PHMap_u8u8*, *num: int = - 1*) → numpy.ndarray[uint64[m, 1]]

rpxdock.phmap.phmap.**test_mod_phmap_inplace** (*arg0: rpxdock.phmap.phmap.PHMap_u8u8*) → None

Module contents

rpxdock.rosetta package

Submodules

rpxdock.rosetta.rosetta_util module

rpxdock.rosetta.rosetta_util.**get_bb_coords** (*pose*, *which_resi=None*)

rpxdock.rosetta.rosetta_util.**get_cb_coords** (*pose*, *which_resi=None*)

rpxdock.rosetta.rosetta_util.**get_sc_coords** (*pose*, *which_resi=None*)

rpxdock.rosetta.rosetta_util.**numpy_stub_from_rosetta_stub** (*rosstub*)

rpxdock.rosetta.rosetta_util.**rosetta_init** (*opts='-beta -mute all'*)

rpxdock.rosetta.triggers_init module

Module contents

rpxdock.rotamer package

Submodules

rpxdock.rotamer.richardson module

rpxdock.rotamer.richardson.**concat_rotamer_space**(*rotspace*)

rpxdock.rotamer.richardson.**get_rotamer_index**(*rotspace*)
extract AA structural info via pyrosetta

rpxdock.rotamer.richardson.**get_rotamer_space**(*concat=False*, *disulf=False*)
B is disulfide, x4 -> x2other n num observations fabo total fraction/alpha/beta/other x#a chi # "act" x# chi # "com" x#r chi # range text x#w chi # peak width lb#/ub# chi # range

rpxdock.rotamer.richardson.**get_rotamer_space_raw**()

rpxdock.rotamer.richardson.**localpath**(*pth*)

rpxdock.rotamer.richardson.**merge_on_chi**(*rs*, *chi*)

rpxdock.rotamer.richardson.**print_full**(*x*)

rpxdock.rotamer.richardson.**sample_rotamer_space**(*rotspace*, *resl=[10, 10, 10, 10]*)
rotamer samples

rpxdock.rotamer.rotamer module

rpxdock.rotamer.rotamer.**assign_rotamers**(*rp*, *rotspace=None*)

rpxdock.rotamer.rotamer.**check_rotamer_deviation**(*rp*, *rotspace*, *quiet=False*)

Module contents

rpxdock.sampling package

Submodules

rpxdock.sampling.compound module

class rpxdock.sampling.compound.**CompoundHier**(**args*)
Bases: object

cell_index_of(*resl*, *idx*)

cellsize(*resl*)

check_indices(*resl*, *idx*)

expand_top_N(*nexpand*, *resl*, *scores*, *indices*)

get_xforms(*resl=0*, *idx=None*)

hier_index_of (*resl*, *idx*)

size (*resl*)

split_indices (*resl*, *idx*)

split_indices_cell (*resl*, *idx*)

split_indices_hier (*resl*, *idx*)

class rpxdock.sampling.compound.**ProductHier** (**args*)
Bases: *rpxdock.sampling.compound.CompoundHier*

combine_xforms (*xparts*)

expand_top_N (*nexpand*, *resl*, *scores*, *indices*)

get_xforms (**args*, ***kw*)

class rpxdock.sampling.compound.**SlideHier** (*sampler*, *body1*, *body2*)
Bases: object

get_xforms (*resl*, *idx*)

class rpxdock.sampling.compound.**ZeroDHier** (*samples*)
Bases: object

cellsize (*resl*)

get_xforms (*resl=0*, *idx=None*)

size (*resl*)

rpxdock.sampling.lattice_hier module

class rpxdock.sampling.lattice_hier.**LatticeHier** (*parts*, *directions*)
Bases: *rpxdock.sampling.compound.CompoundHier*

combine_xforms (*xparts*)

expand_top_N (*nexpand*, *resl*, *scores*, *indices*)

get_xforms (**args*, ***kw*)

rpxdock.sampling.orientations module

rpxdock.sampling.orientations.**filter_quaternion_set_axis_within** (*quats*, *axis*, *angle*)

rpxdock.sampling.orientations.**karney_data_path** (*fname*)

rpxdock.sampling.orientations.**karney_name_by_radius** (*cr*)

rpxdock.sampling.orientations.**quaternion_set_by_name** (*name*)

rpxdock.sampling.orientations.**quaternion_set_with_covering_radius_degrees** (*cr=63*)

rpxdock.sampling.orientations.**quats_from_karney_file** (*fname*)

rpxdock.sampling.sphere module

`rpxdock.sampling.sphere.`**`get_sphere_samples`**(*sym=None*)

rpxdock.sampling.xform_grid module

`rpxdock.sampling.xform_grid.`**`grid_sym_axis`**(*cart, ang, axis=[0, 0, 1], flip=None*)

rpxdock.sampling.xform_hier module

`class` `rpxdock.sampling.xform_hier.`**`LineHier`**(*lb, ub, nstep, axis*)
Bases: `object`

`get_xforms`(*resl, idx*)

`rpxdock.sampling.xform_hier.`**`hier_axis_sampler`**(*nfold, lb=25, ub=200, resl=10, angresl=10, axis=[0, 0, 1], flipax=[0, 1, 0]*)

`rpxdock.sampling.xform_hier.`**`hier_mirror_lattice_sampler`**(*spec, cart_bounds=[0, 100], resl=10, angresl=10, flip_components=True, **kw*)

`rpxdock.sampling.xform_hier.`**`hier_multi_axis_sampler`**(*spec, cart_bounds=[25, 200], resl=10, angresl=10, flip_components=True, **kw*)

rpxdock.sampling.xform_hierarchy module

`class` `rpxdock.sampling.xform_hierarchy.`**`CartHier1D_f4`**
Bases: `pybind11_builtins.pybind11_object`

`property dim`

`get_trans`(*self:* rpxdock.sampling.xform_hierarchy.CartHier1D_f4, *arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

`get_xforms`(*self:* rpxdock.sampling.xform_hierarchy.CartHier1D_f4, *arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

`property ncell`

`sanity_check`(*self:* rpxdock.sampling.xform_hierarchy.CartHier1D_f4) → bool

`size`(*self:* rpxdock.sampling.xform_hierarchy.CartHier1D_f4, *arg0: int*) → int

`class` `rpxdock.sampling.xform_hierarchy.`**`CartHier1D_f8`**
Bases: `pybind11_builtins.pybind11_object`

`property dim`

`get_trans`(*self:* rpxdock.sampling.xform_hierarchy.CartHier1D_f8, *arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

`get_xforms`(*self:* rpxdock.sampling.xform_hierarchy.CartHier1D_f8, *arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

`property ncell`

`sanity_check`(*self:* rpxdock.sampling.xform_hierarchy.CartHier1D_f8) → bool

size (*self: rpxdock.sampling.xform_hierarchy.CartHier1D_f8, arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**CartHier2D_f4**
Bases: pybind11_builtins.pybind11_object

property dim

get_trans (*self: rpxdock.sampling.xform_hierarchy.CartHier2D_f4, arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.CartHier2D_f4, arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self: rpxdock.sampling.xform_hierarchy.CartHier2D_f4*) → bool

size (*self: rpxdock.sampling.xform_hierarchy.CartHier2D_f4, arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**CartHier2D_f8**
Bases: pybind11_builtins.pybind11_object

property dim

get_trans (*self: rpxdock.sampling.xform_hierarchy.CartHier2D_f8, arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.CartHier2D_f8, arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self: rpxdock.sampling.xform_hierarchy.CartHier2D_f8*) → bool

size (*self: rpxdock.sampling.xform_hierarchy.CartHier2D_f8, arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**CartHier3D_f4**
Bases: pybind11_builtins.pybind11_object

property dim

get_trans (*self: rpxdock.sampling.xform_hierarchy.CartHier3D_f4, arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.CartHier3D_f4, arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self: rpxdock.sampling.xform_hierarchy.CartHier3D_f4*) → bool

size (*self: rpxdock.sampling.xform_hierarchy.CartHier3D_f4, arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**CartHier3D_f8**
Bases: pybind11_builtins.pybind11_object

property dim

get_trans (*self: rpxdock.sampling.xform_hierarchy.CartHier3D_f8, arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.CartHier3D_f8, arg0: int, arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self: rpxdock.sampling.xform_hierarchy.CartHier3D_f8*) → bool

size (*self: rpxdock.sampling.xform_hierarchy.CartHier3D_f8, arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**CartHier4D_f4**
Bases: pybind11_builtins.pybind11_object

property dim

get_trans (*self:* rpxdock.sampling.xform_hierarchy.CartHier4D_f4, *arg0:* *int*, *arg1:* *numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self:* rpxdock.sampling.xform_hierarchy.CartHier4D_f4, *arg0:* *int*, *arg1:* *numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self:* rpxdock.sampling.xform_hierarchy.CartHier4D_f4) → bool

size (*self:* rpxdock.sampling.xform_hierarchy.CartHier4D_f4, *arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**CartHier4D_f8**
Bases: pybind11_builtins.pybind11_object

property dim

get_trans (*self:* rpxdock.sampling.xform_hierarchy.CartHier4D_f8, *arg0:* *int*, *arg1:* *numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self:* rpxdock.sampling.xform_hierarchy.CartHier4D_f8, *arg0:* *int*, *arg1:* *numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self:* rpxdock.sampling.xform_hierarchy.CartHier4D_f8) → bool

size (*self:* rpxdock.sampling.xform_hierarchy.CartHier4D_f8, *arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**CartHier5D_f4**
Bases: pybind11_builtins.pybind11_object

property dim

get_trans (*self:* rpxdock.sampling.xform_hierarchy.CartHier5D_f4, *arg0:* *int*, *arg1:* *numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self:* rpxdock.sampling.xform_hierarchy.CartHier5D_f4, *arg0:* *int*, *arg1:* *numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self:* rpxdock.sampling.xform_hierarchy.CartHier5D_f4) → bool

size (*self:* rpxdock.sampling.xform_hierarchy.CartHier5D_f4, *arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**CartHier5D_f8**
Bases: pybind11_builtins.pybind11_object

property dim

get_trans (*self:* rpxdock.sampling.xform_hierarchy.CartHier5D_f8, *arg0:* *int*, *arg1:* *numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self:* rpxdock.sampling.xform_hierarchy.CartHier5D_f8, *arg0:* *int*, *arg1:* *numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self:* rpxdock.sampling.xform_hierarchy.CartHier5D_f8) → bool

size (*self:* rpxdock.sampling.xform_hierarchy.CartHier5D_f8, *arg0: int*) → int

class `rpxdock.sampling.xform_hierarchy.`**`CartHier6D_f4`**
Bases: `pybind11_builtins.pybind11_object`

property dim

get_trans (*self: rpxdock.sampling.xform_hierarchy.CartHier6D_f4*, *arg0: int*, *arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.CartHier6D_f4*, *arg0: int*, *arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self: rpxdock.sampling.xform_hierarchy.CartHier6D_f4*) → bool

size (*self: rpxdock.sampling.xform_hierarchy.CartHier6D_f4*, *arg0: int*) → int

class `rpxdock.sampling.xform_hierarchy.`**`CartHier6D_f8`**
Bases: `pybind11_builtins.pybind11_object`

property dim

get_trans (*self: rpxdock.sampling.xform_hierarchy.CartHier6D_f8*, *arg0: int*, *arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.CartHier6D_f8*, *arg0: int*, *arg1: numpy.ndarray[uint64[m, 1]]*) → tuple

property ncell

sanity_check (*self: rpxdock.sampling.xform_hierarchy.CartHier6D_f8*) → bool

size (*self: rpxdock.sampling.xform_hierarchy.CartHier6D_f8*, *arg0: int*) → int

class `rpxdock.sampling.xform_hierarchy.`**`DummyHier_f4`**
Bases: `pybind11_builtins.pybind11_object`

child_of_begin (*self: rpxdock.sampling.xform_hierarchy.DummyHier_f4*, *arg0: int*) → int

child_of_end (*self: rpxdock.sampling.xform_hierarchy.DummyHier_f4*, *arg0: int*) → int

expand_top_N (*self: rpxdock.sampling.xform_hierarchy.DummyHier_f4*, *nkeep: int*, *resl: int*, *score: numpy.ndarray[float64[m, 1]]*, *index: numpy.ndarray[uint64[m, 1]]*, *null_val: float = 0*) → tuple

size (*self: rpxdock.sampling.xform_hierarchy.DummyHier_f4*, *arg0: int*) → int

class `rpxdock.sampling.xform_hierarchy.`**`DummyHier_f8`**
Bases: `pybind11_builtins.pybind11_object`

child_of_begin (*self: rpxdock.sampling.xform_hierarchy.DummyHier_f8*, *arg0: int*) → int

child_of_end (*self: rpxdock.sampling.xform_hierarchy.DummyHier_f8*, *arg0: int*) → int

expand_top_N (*self: rpxdock.sampling.xform_hierarchy.DummyHier_f8*, *nkeep: int*, *resl: int*, *score: numpy.ndarray[float64[m, 1]]*, *index: numpy.ndarray[uint64[m, 1]]*, *null_val: float = 0*) → tuple

size (*self: rpxdock.sampling.xform_hierarchy.DummyHier_f8*, *arg0: int*) → int

class `rpxdock.sampling.xform_hierarchy.`**`OriCart1Hier_f4`**
Bases: `pybind11_builtins.pybind11_object`

property cart_bs

property cart_cell_width

property cart_lb

property cart_ncell

property cart_ub

cell_index_of(*self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4*, *arg0: numpy.ndarray[uint64]*, *arg1: numpy.ndarray[uint64]*) → object

child_of_begin(*self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4*, *arg0: numpy.ndarray[uint64]*) → object

child_of_end(*self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4*, *arg0: numpy.ndarray[uint64]*) → object

property dim

expand_top_N(**args*, ***kwargs*)
Overloaded function.

1. expand_top_N(self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4, nkeep: int, resl: int, score_idx: numpy.ndarray[rpxdock::sampling::ScoreIndex], null_val: float = 0) -> tuple
2. expand_top_N(self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4, nkeep: int, resl: int, score: numpy.ndarray[float64[m, 1]], index: numpy.ndarray[uint64[m, 1]], null_val: float = 0) -> tuple

get_xforms(*self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4*, *iresl: int*, *idx: numpy.ndarray[uint64[m, 1]]*) → tuple

hier_index_of(*self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4*, *arg0: numpy.ndarray[uint64]*, *arg1: numpy.ndarray[uint64]*) → object

property ncell

property ori_ncell

property ori_nside

property ori_resl

parent_of(*self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4*, *arg0: numpy.ndarray[uint64]*) → object

sanity_check(*self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4*) → bool

size(*self: rpxdock.sampling.xform_hierarchy.OriCart1Hier_f4*, *arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**OriHier_f4**
Bases: pybind11_builtins.pybind11_object

get_ori(**args*, ***kwargs*)
Overloaded function.

1. get_ori(self: rpxdock.sampling.xform_hierarchy.OriHier_f4, resl: int, idx: numpy.ndarray[uint64[m, 1]]) -> tuple
2. get_ori(self: rpxdock.sampling.xform_hierarchy.OriHier_f4, resl: int, idx: numpy.ndarray[uint64[m, 1]]) -> tuple

property ncell

property ori_nside

size(*self: rpxdock.sampling.xform_hierarchy.OriHier_f4*, *arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**OriHier_f8**
Bases: pybind11_builtins.pybind11_object

get_ori(**args*, ***kwargs*)
Overloaded function.

1. get_ori(self: rpxdock.sampling.xform_hierarchy.OriHier_f8, resl: int, idx: numpy.ndarray[uint64[m, 1]]) -> tuple
2. get_ori(self: rpxdock.sampling.xform_hierarchy.OriHier_f8, resl: int, idx: numpy.ndarray[uint64[m, 1]]) -> tuple

property ncell

property ori_nside

size (*self:* rpxdock.sampling.xform_hierarchy.OriHier_f8, *arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**RotCart1Hier_f4**
Bases: pybind11_builtins.pybind11_object

property axis

property cart_bs_

property cart_bs_pref_prod_

property cart_cell_width_

property cart_lb

property cart_ncell

property cart_ub

cell_index_of (*self:* rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4, *arg0: numpy.ndarray[uint64]*, *arg1: numpy.ndarray[uint64]*) → object

child_of_begin (*self:* rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4, *arg0: numpy.ndarray[uint64]*) → object

child_of_end (*self:* rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4, *arg0: numpy.ndarray[uint64]*) → object

property dim

expand_top_N (**args*, ***kwargs*)
Overloaded function.

1. expand_top_N(self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4, nkeep: int, resl: int, score_idx: numpy.ndarray[rpxdock::sampling::ScoreIndex], null_val: float = 0) -> tuple
2. expand_top_N(self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4, nkeep: int, resl: int, score: numpy.ndarray[float64[m, 1]], index: numpy.ndarray[uint64[m, 1]], null_val: float = 0) -> tuple

get_xforms (*self:* rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4, *iresl: int*, *idx: numpy.ndarray[uint64[m, 1]]*) → tuple

hier_index_of (*self:* rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4, *arg0: numpy.ndarray[uint64]*, *arg1: numpy.ndarray[uint64]*) → object

property ncell

parent_of (*self:* rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4, *arg0: numpy.ndarray[uint64]*) → object

property rot_cell_width

property rot_lb

property rot_ncell

property rot_ub

sanity_check (*self:* rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4) → bool

size (*self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f4*, *arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**RotCart1Hier_f8**

Bases: pybind11_builtins.pybind11_object

property axis

property cart_bs_

property cart_bs_pref_prod_

property cart_cell_width_

property cart_lb

property cart_ncell

property cart_ub

cell_index_of (*self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8*, *arg0: numpy.ndarray[uint64]*, *arg1: numpy.ndarray[uint64]*) → object

child_of_begin (*self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8*, *arg0: numpy.ndarray[uint64]*) → object

child_of_end (*self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8*, *arg0: numpy.ndarray[uint64]*) → object

property dim

expand_top_N (**args*, ***kwargs*)

Overloaded function.

1. expand_top_N(self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8, nkeep: int, resl: int, score_idx: numpy.ndarray[rpxdock::sampling::ScoreIndex], null_val: float = 0) -> tuple
2. expand_top_N(self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8, nkeep: int, resl: int, score: numpy.ndarray[float64[m, 1]], index: numpy.ndarray[uint64[m, 1]], null_val: float = 0) -> tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8*, *iresl: int*, *idx: numpy.ndarray[uint64[m, 1]]*) → tuple

hier_index_of (*self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8*, *arg0: numpy.ndarray[uint64]*, *arg1: numpy.ndarray[uint64]*) → object

property ncell

parent_of (*self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8*, *arg0: numpy.ndarray[uint64]*) → object

property rot_cell_width

property rot_lb

property rot_ncell

property rot_ub

sanity_check (*self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8*) → bool

size (*self: rpxdock.sampling.xform_hierarchy.RotCart1Hier_f8*, *arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**RotHier_f4**

Bases: pybind11_builtins.pybind11_object

property axis

property dim

get_ori (*self: rpxdock.sampling.xform_hierarchy.RotHier_f4*, *resl: int*, *idx: numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.RotHier_f4*, *resl: int*, *idx: numpy.ndarray[uint64[m, 1]]*) → tuple

property lb

property ncell

property rot_cell_width

property rot_ncell

size (*self: rpxdock.sampling.xform_hierarchy.RotHier_f4*, *arg0: int*) → int

property ub

class rpxdock.sampling.xform_hierarchy.**RotHier_f8**

Bases: `pybind11_builtins.pybind11_object`

property axis

property dim

get_ori (*self: rpxdock.sampling.xform_hierarchy.RotHier_f8*, *resl: int*, *idx: numpy.ndarray[uint64[m, 1]]*) → tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.RotHier_f8*, *resl: int*, *idx: numpy.ndarray[uint64[m, 1]]*) → tuple

property lb

property ncell

property rot_cell_width

property rot_ncell

size (*self: rpxdock.sampling.xform_hierarchy.RotHier_f8*, *arg0: int*) → int

property ub

class rpxdock.sampling.xform_hierarchy.**XformHier_f4**

Bases: `pybind11_builtins.pybind11_object`

property cart_bs

property cart_cell_width

property cart_lb

property cart_ncell

property cart_ub

cell_index_of (*self: rpxdock.sampling.xform_hierarchy.XformHier_f4*, *arg0: numpy.ndarray[uint64]*, *arg1: numpy.ndarray[uint64]*) → object

child_of_begin (*self: rpxdock.sampling.xform_hierarchy.XformHier_f4*, *arg0: numpy.ndarray[uint64]*) → object

child_of_end (*self: rpxdock.sampling.xform_hierarchy.XformHier_f4*, *arg0: numpy.ndarray[uint64]*) → object

property dim

expand_top_N (**args*, ***kwargs*)

Overloaded function.

1. expand_top_N(self: rpxdock.sampling.xform_hierarchy.XformHier_f4, nkeep: int, resl: int, score_idx: numpy.ndarray[rpxdock::sampling::ScoreIndex], null_val: float = 0) -> tuple
2. expand_top_N(self: rpxdock.sampling.xform_hierarchy.XformHier_f4, nkeep: int, resl: int, score: numpy.ndarray[float64[m, 1]], index: numpy.ndarray[uint64[m, 1]], null_val: float = 0) -> tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.XformHier_f4, iresl: int, idx: numpy.ndarray[uint64[m, 1]]*) → tuple

hier_index_of (*self: rpxdock.sampling.xform_hierarchy.XformHier_f4, arg0: numpy.ndarray[uint64], arg1: numpy.ndarray[uint64]*) → object

property ncell

property ori_ncell

property ori_nside

property ori_resl

parent_of (*self: rpxdock.sampling.xform_hierarchy.XformHier_f4, arg0: numpy.ndarray[uint64]*) → object

sanity_check (*self: rpxdock.sampling.xform_hierarchy.XformHier_f4*) → bool

size (*self: rpxdock.sampling.xform_hierarchy.XformHier_f4, arg0: int*) → int

class rpxdock.sampling.xform_hierarchy.**XformHier_f8**

Bases: pybind11_builtins.pybind11_object

property cart_bs

property cart_cell_width

property cart_lb

property cart_ncell

property cart_ub

cell_index_of (*self: rpxdock.sampling.xform_hierarchy.XformHier_f8, arg0: numpy.ndarray[uint64], arg1: numpy.ndarray[uint64]*) → object

child_of_begin (*self: rpxdock.sampling.xform_hierarchy.XformHier_f8, arg0: numpy.ndarray[uint64]*) → object

child_of_end (*self: rpxdock.sampling.xform_hierarchy.XformHier_f8, arg0: numpy.ndarray[uint64]*) → object

property dim

expand_top_N (**args, **kwargs*)

Overloaded function.

1. expand_top_N(self: rpxdock.sampling.xform_hierarchy.XformHier_f8, nkeep: int, resl: int, score_idx: numpy.ndarray[rpxdock::sampling::ScoreIndex], null_val: float = 0) -> tuple
2. expand_top_N(self: rpxdock.sampling.xform_hierarchy.XformHier_f8, nkeep: int, resl: int, score: numpy.ndarray[float64[m, 1]], index: numpy.ndarray[uint64[m, 1]], null_val: float = 0) -> tuple

get_xforms (*self: rpxdock.sampling.xform_hierarchy.XformHier_f8, iresl: int, idx: numpy.ndarray[uint64[m, 1]]*) → tuple

hier_index_of (*self: rpxdock.sampling.xform_hierarchy.XformHier_f8, arg0: numpy.ndarray[uint64], arg1: numpy.ndarray[uint64]*) → object

property ncell

property ori_ncell

property ori_nside

property ori_resl

parent_of (*self:* rpxdock.sampling.xform_hierarchy.XformHier_f8, *arg0: numpy.ndarray[uint64]*) → object

sanity_check (*self:* rpxdock.sampling.xform_hierarchy.XformHier_f8) → bool

size (*self:* rpxdock.sampling.xform_hierarchy.XformHier_f8, *arg0: int*) → int

rpxdock.sampling.xform_hierarchy.**coeffs3zorder** (*arg0: numpy.ndarray[uint64[m, 4], flags.writeable, flags.c_contiguous], arg1: int*) → numpy.ndarray[uint64[m, 1]]

rpxdock.sampling.xform_hierarchy.**coeffs6zorder** (*arg0: numpy.ndarray[uint64[m, 7], flags.writeable, flags.c_contiguous], arg1: int*) → numpy.ndarray[uint64[m, 1]]

rpxdock.sampling.xform_hierarchy.**create_OriCart1Hier_4f_nside** (*lb: numpy.ndarray[float32[1, 1]], ub: numpy.ndarray[float32[1, 1]], bs: numpy.ndarray[uint64[1, 1]], nside: int*) → *rpxdock.sampling.xform_hierarchy.OriCart1H*

rpxdock.sampling.xform_hierarchy.**create_OriHier_nside_f4** (*nside: int*) → *rpxdock.sampling.xform_hierarchy.OriHier_f4*

rpxdock.sampling.xform_hierarchy.**create_OriHier_nside_f8** (*nside: int*) → *rpxdock.sampling.xform_hierarchy.OriHier_f8*

rpxdock.sampling.xform_hierarchy.**create_RotHier_nside_f4** (*lb: float, ub: float, ncell: int, axis: numpy.ndarray[float32[3, 1]] = array([0.0, 0.0, 1.0], dtype=float32)*) → *rpxdock.sampling.xform_hierarchy.RotHier_f4*

rpxdock.sampling.xform_hierarchy.**create_RotHier_nside_f8** (*lb: float, ub: float, ncell: int, axis: numpy.ndarray[float64[3, 1]] = array([0.0, 0.0, 1.0])*) → *rpxdock.sampling.xform_hierarchy.RotHier_f8*

rpxdock.sampling.xform_hierarchy.**create_XformHier_nside_f4** (*lb: numpy.ndarray[float32[3, 1]], ub: numpy.ndarray[float32[3, 1]], bs: numpy.ndarray[uint64[3, 1]], nside: int*) → *rpxdock.sampling.xform_hierarchy.XformHier_f4* `rpxdock.sampling.xform_hierarchy.`**`create_XformHier_nside_f8`**(*lb: numpy.ndarray[float64[3, 1]], ub: numpy.ndarray[float64[3, 1]], bs: numpy.ndarray[uint64[3, 1]], nside: int*) → *rpxdock.sampling.xform_hierarchy.XformHier_f8*

`rpxdock.sampling.xform_hierarchy.`**`pack_zorder`**(*resl: int, indices: numpy.ndarray[uint64[m, n]]*) → numpy.ndarray[uint64[m, 1]]

`rpxdock.sampling.xform_hierarchy.`**`unpack_zorder`**(*dim: int, resl: int, indices: numpy.ndarray[uint64[m, 1]]*) → numpy.ndarray[uint64[m, n]]

`rpxdock.sampling.xform_hierarchy.`**`zorder3coeffs`**(*arg0: numpy.ndarray[uint64[m, 1]], arg1: int*) → numpy.ndarray[uint64[m, 4]]

`rpxdock.sampling.xform_hierarchy.`**`zorder6coeffs`**(*arg0: numpy.ndarray[uint64[m, 1]], arg1: int*) → numpy.ndarray[uint64[m, 7]]

rpxdock.sampling.xhier_util module

`rpxdock.sampling.xhier_util.`**`xform_hier_guess_sampling_covrads`**(*hierarchy_depth, sampling_lever, base_sample_resl, base_cart_resl, base_ori_resl, xhier_cart_fudge_factor, xhier_ori_fudge_factor, **kw*)

Module contents

rpxdock.score package

Submodules

rpxdock.score.body_pair_eval module

class `rpxdock.score.body_pair_eval.`**`BodyPairEvalComponent`**(*name=None, priority: float = 0*)

Bases: `abc.ABC`

`check_validity`()

`extra_fields = []`

`score_fields = []`

class `rpxdock.score.body_pair_eval.`**`BodyPairEvaluator`**(*components*)

Bases: `object`

rpxdock.score.rpxhier module

class rpxdock.score.rpxhier.**RpxHier**(*files*, *max_pair_dist=8.0*, *hscore_data_dir=None*, ***kw*)
Bases: object

hier_mindis(*iresl*)

iresls()

score(*body1*, *body2*, *wts*, *iresl=- 1*, **bounds*)

score_all(*x*)

score_base(*x_or_k*)

score_by_resl(*resl*, *x_or_k*)

score_matrix_inter(*bodyA*, *bodyB*, *wts*, *symframes=[array([[1.0, 0.0, 0.0, 0.0], [0.0, 1.0, 0.0, 0.0], [0.0, 0.0, 1.0, 0.0], [0.0, 0.0, 0.0, 1.0]])]*, *iresl=- 1*, ***kw*)

score_matrix_intra(*body*, *wts*, *iresl=- 1*)

scorepos(*body1*, *body2*, *pos1*, *pos2*, *iresl=- 1*, *bounds=()*, ***kw*)

Module contents

rpxdock.search package

Submodules

rpxdock.search.asym module

class rpxdock.search.asym.**AsymEvaluator**(*bodies*, *hscore*, ***kw*)
Bases: object

rpxdock.search.asym.**asym_get_sample_hierarchy**(*body*, *hscore*, *extent=100*)
set up XformHier with appropriate bounds and resolution

rpxdock.search.asym.**make_asym**(*bodies*, *hscore*, *sampler*, *search=<function hier_search>*, ***kw*)

rpxdock.search.basic module

rpxdock.search.basic.**evaluate_positions**(*evaluator*, *xforms*, *executor=None*, ***kw*)

rpxdock.search.basic.**evaluate_positions_executor**(*executor*, *evaluator*, *xforms*, ***kw*)

rpxdock.search.basic.**grid_search**(*sampler*, *evaluator*, ***kw*)

rpxdock.search.basic.**trim_ok**(*trim*, *nres*, *max_trim*, ***kw*)

rpxdock.search.cyclic module

class rpxdock.search.cyclic.**CyclicEvaluator**(*body*, *sym*, *hscore*, ***kw*)
Bases: object

rpxdock.search.cyclic.**make_cyclic**(*monomer*, *sym*, *hscore*, *search=<function hier_search>*, *sampler=None*, ***kw*)

rpxdock.search.cyclic.**make_cyclic_hier_sampler**(*monomer*, *hscore*)

rpxdock.search.dockspec module

class rpxdock.search.dockspec.**DockSpec1CompCage**(*arch*)
Bases: object

place_along_axis(*pos*, *slide_dist*)

placements(*angles*)

placements_second(*placements*)

slide_dir()

symframes(*cellspacing=None*, *radius=None*)

class rpxdock.search.dockspec.**DockSpec1CompMirrorLayer**(*arch*)
Bases: object

class rpxdock.search.dockspec.**DockSpec2CompCage**(*arch*)
Bases: object

move_to_canonical_unit(*pos1*, *pos2*)

place_along_axes(*pos1*, *pos2*)

placements1(*angles*, *flip=True*)

placements2(*angles*)

slide_dir(*angles*)

symframes(*cellspacing=None*, *radius=None*)

class rpxdock.search.dockspec.**DockSpec3CompCage**(*arch*)
Bases: object

class rpxdock.search.dockspec.**DockSpec3CompLayer**(*arch*)
Bases: object

class rpxdock.search.dockspec.**DockSpecMonomerToCyclic**(*arch*)
Bases: object

place_along_axis(*pos*, *slide_dist*)

placements_second(*placements*)

slide_dir()

symframes(*cellspacing=None*, *radius=None*)

rpxdock.search.gridslide module

rpxdock.search.gridslide.**find_connected_1xCyclic_slide**(*spec*, *body*, *samples*, *min_contacts=30*, *contact_dis=8.0*)

rpxdock.search.gridslide.**find_connected_2xCyclic_slide**(*spec*, *body1*, *body2*, *samples*, *min_contacts=30*, *contact_dis=8.0*, *onebody=True*, ***kw*)

rpxdock.search.gridslide.**find_connected_monomer_to_cyclic_slide**(*spec*, *body*, *samples*, *min_contacts*, *contact_dis*)

rpxdock.search.gridslide.**samples_1xCyclic**(*spec*, *resl=1*)

rpxdock.search.gridslide.**samples_1xMonomer_orientations**(*resl*)

rpxdock.search.gridslide.**samples_2xCyclic_slide**(*spec*, *resl=1*, *max_out_of_plane_angle=10*, ***kw*)

rpxdock.search.helix module

class rpxdock.search.helix.**HelixEvaluator**(*body*, *hscore*, ***kw*)

Bases: object

rpxdock.search.helix.**helix_get_sample_hierarchy**(*body*, *hscore*, *extent=100*)

set up XformHier with appropriate bounds and resolution

rpxdock.search.helix.**make_helix**(*body*, *hscore*, *sampler*, *search=<function hier_search>*, ***kw*)

rpxdock.search.hierarchical module

rpxdock.search.hierarchical.**expand_samples**(*iresl*, *sampler*, *indices=None*, *scores=None*, *beam_size=None*, ***kw*)

rpxdock.search.hierarchical.**hier_search**(*sampler*, *evaluator*, ***kw*)

rpxdock.search.multicomp module

class rpxdock.search.multicomp.**MultiCompEvaluator**(**arg*, ***kw*)

Bases: *rpxdock.search.multicomp.MultiCompEvaluatorBase*

class rpxdock.search.multicomp.**MultiCompEvaluatorBase**(*bodies*, *spec*, *hscore*, *wts={'ncontact': 0.1, 'rpx': 1.0}*, ***kw*)

Bases: object

class rpxdock.search.multicomp.**TwoCompEvaluatorWithTrim**(**arg*, *trimmable_components='AB'*, ***kw*)

Bases: *rpxdock.search.multicomp.MultiCompEvaluatorBase*

eval_trim_one(*trim_component*, *x*, *iresl=- 1*, *wts={}*, ***kw*) rpxdock.search.multicomp.**make_multicomp**(*bodies*, *spec*, *hscore*, *search=<function hier_search>*, *sampler=None*, *fixed_components=False*, ***kw*)

rpxdock.search.onecomp module

class rpxdock.search.onecomp.**OneCompEvaluator**(*body*, *spec*, *hscore*, *wts={'ncontact': 0.1, 'rpx': 1.0}*, *trimmable_components='AB'*, ***kw*)

Bases: object

rpxdock.search.onecomp.**make_onecomp**(*body*, *spec*, *hscore*, *search=<function hier_search>*, *sampler=None*, *fixed_components=False*, ***kw*)

rpxdock.search.plug module

class rpxdock.search.plug.**PlugEvaluator**(*plug*, *hole*, *hscore*, ***kw*)

Bases: object

iface_scores(*xforms*, *iresl=- 1*, *wts={}*, ***_*)

rpxdock.search.plug.**make_plugs**(*plug*, *hole*, *hscore*, *search=<function hier_search>*, *sampler=None*, ***kw*)

rpxdock.search.plug.**plug_get_sample_hierarchy**(*plug*, *hole*, *hscore*)

set up XformHier with appropriate bounds and resolution

rpxdock.search.plug.**plug_test_hier_sampler**(*plug*, *hole*, *hscore*, *n=6*)

rpxdock.search.result module

class rpxdock.search.result.**Result**(*data_or_file=None*, *body_=[]*, *body_label_=None*, ***kw*)

Bases: object

copy()

dump_pdb(*imodel*, *output_prefix=''*, *output_suffix=''*, *fname=None*, *output_body='ALL'*, *sym=''*, *sep='_'*, *skip=[]*, *hscore=None*, *output_asym_only=False*, ***kw*)

dump_pdbs(*which*, *ndigwhich=None*, *ndigmdl=None*, *lbl=''*, *skip=[]*, *output_prefix='rpx'*, ***kw*)

dump_pdbs_top_score(*nout_top=10*, ***kw*)

dump_pdbs_top_score_each(*nout_each=1*, ***kw*)

getstate()

property ndocks

sel(**args*, ***kw*)

setstate(*state*)

sortby(**args*, ***kw*)

top_each(*neach=1*)

rpxdock.search.result.**assert_results_close**(*r*, *s*, *n=- 1*)

rpxdock.search.result.**concat_results**(*results*, ***kw*) rpxdock.search.result.**dict_coherent_entries**(*alldicts*)

rpxdock.search.result.**dummy_result**(*size=1000*)

Module contents

rpxdock.tests package

Subpackages

rpxdock.tests.app package

Submodules

rpxdock.tests.app.test_options module

rpxdock.tests.app.test_options.**test_defaults**()

rpxdock.tests.app.test_options.**test_get_cli_args**()

Module contents

rpxdock.tests.bvh package

Submodules

rpxdock.tests.bvh.test_bvh module

rpxdock.tests.bvh.test_bvh.**random_walk**(*N*)

rpxdock.tests.bvh.test_bvh.**test_bvh_accessors**()

rpxdock.tests.bvh.test_bvh.**test_bvh_isect**()

rpxdock.tests.bvh.test_bvh.**test_bvh_isect_cpp**()

rpxdock.tests.bvh.test_bvh.**test_bvh_isect_fixed**()

rpxdock.tests.bvh.test_bvh.**test_bvh_isect_fixed_range**()

rpxdock.tests.bvh.test_bvh.**test_bvh_isect_range**(*body=None, cart_sd=0.3, N2=10, mindist=0.02*)

rpxdock.tests.bvh.test_bvh.**test_bvh_isect_range_ids**()

rpxdock.tests.bvh.test_bvh.**test_bvh_isect_range_lb_ub**(*body=None, cart_sd=0.3, N1=3, N2=20, mindist=0.02*)

rpxdock.tests.bvh.test_bvh.**test_bvh_min_cpp**()

rpxdock.tests.bvh.test_bvh.**test_bvh_min_dist**()

rpxdock.tests.bvh.test_bvh.**test_bvh_min_dist_fixed**()

rpxdock.tests.bvh.test_bvh.**test_bvh_min_dist_floormin**()

rpxdock.tests.bvh.test_bvh.**test_bvh_pickle**(*tmpdir*) rpxdock.tests.bvh.test_bvh.**test_bvh_slide_single**()

rpxdock.tests.bvh.test_bvh.**test_bvh_slide_single_inline**()

rpxdock.tests.bvh.test_bvh.**test_bvh_slide_single_xform**()

rpxdock.tests.bvh.test_bvh.**test_bvh_slide_whole**()

rpxdock.tests.bvh.test_bvh.**test_bvh_threading_isect_may_fail**()

rpxdock.tests.bvh.test_bvh.**test_bvh_threading_mindist_may_fail**()

rpxdock.tests.bvh.test_bvh.**test_collect_pairs**()

rpxdock.tests.bvh.test_bvh.**test_collect_pairs_range**()

rpxdock.tests.bvh.test_bvh.**test_collect_pairs_range_sym**()

rpxdock.tests.bvh.test_bvh.**test_collect_pairs_simple**()

rpxdock.tests.bvh.test_bvh.**test_collect_pairs_simple_selection**()

rpxdock.tests.bvh.test_bvh.**test_slide_collect_pairs**()

rpxdock.tests.bvh.test_bvh_nd module

rpxdock.tests.bvh.test_bvh_nd.**test_bvh_bvh_isect7**()

rpxdock.tests.bvh.test_bvh_nd.**test_bvh_isect7**()

rpxdock.tests.bvh.test_bvh_nd.**test_bvh_mindist4**()

rpxdock.tests.bvh.test_bvh_nd.**test_bvh_mindist7**()

Module contents

rpxdock.tests.cluster package

Submodules

rpxdock.tests.cluster.test_cluster module

rpxdock.tests.cluster.test_cluster.**test_cluster**()

rpxdock.tests.cluster.test_cluster.**test_cluster_rand**()

Module contents

rpxdock.tests.geom package

Submodules

rpxdock.tests.geom.test_bcc module

rpxdock.tests.geom.test_bcc.**test_bcc_neighbor_radous**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_3**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_3_exhalf**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_3_exhalf_sphere**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_3_sphere**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_6_3**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_6_3_extrahalf**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_6_3_oddlast3**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_6_3_oddlast3_extrahalf**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_6_3_oddlast3_sphere**()

rpxdock.tests.geom.test_bcc.**test_bcc_neighbors_6_3_oddlast3_sphere_extrahalf**()

rpxdock.tests.geom.test_geom module

rpxdock.tests.geom.test_geom.**test_geom_sphere**()

rpxdock.tests.geom.test_geom.**test_geom_welzl_sphere**()

rpxdock.tests.geom.test_geom.**test_miniball_cpp**()

rpxdock.tests.geom.test_geom.**test_miniball_py**()

rpxdock.tests.geom.test_geom.**test_xform_dist**()

rpxdock.tests.geom.test_quat module

rpxdock.tests.geom.test_quat.**test_quat_mult**()

rpxdock.tests.geom.test_quat.**test_rand_quat**()

rpxdock.tests.geom.test_quat.**test_rot_quat_conversion_cases**()

rpxdock.tests.geom.test_quat.**test_rot_quat_conversion_rand**()

rpxdock.tests.geom.test_sym module

rpxdock.tests.geom.test_sym.**test_sym**()

rpxdock.tests.geom.test_sym.**test_symframes**()

Module contents

rpxdock.tests.motif package

Submodules

rpxdock.tests.motif.test_motif module

rpxdock.tests.motif.test_motif.**get_pair_keys_ss_py**(*rp*, *xbin*, *min_ssep*)

rpxdock.tests.motif.test_motif.**pair_key_ss_py**(*xbin*, *resi*, *resj*, *ss*, *stub*) rpxdock.tests.motif.test_motif.**test_jagged_bin**()

rpxdock.tests.motif.test_motif.**test_jagged_bin_zero**()

rpxdock.tests.motif.test_motif.**test_pair_key**(*respairdat*)

rpxdock.tests.motif.test_motif.**test_pair_key_ss**(*respairdat*)

rpxdock.tests.motif.test_motif.**test_remove_redundant_pdbs**()

rpxdock.tests.motif.test_motif.**test_respairdat_addrots**(*respairdat*)

rpxdock.tests.motif.test_pairdat module

rpxdock.tests.motif.test_pairdat.**test_pairdat_subset_by_aa**(*respairdat*)

rpxdock.tests.motif.test_pairdat.**test_pairdat_subset_by_ss**(*respairdat*)

rpxdock.tests.motif.test_pairdat.**test_tiny_subset_by_aa_pdb_removal_2pdb**()

rpxdock.tests.motif.test_pairdat.**test_tiny_subset_by_aa_pdb_removal_3pdb**()

rpxdock.tests.motif.test_pairscore module

rpxdock.tests.motif.test_pairscore.**make_score_files_moveme**()

rpxdock.tests.motif.test_pairscore.**terrible_sanity_check**()

rpxdock.tests.motif.test_pairscore.**test_bin_get_all_data**(*respairscore*)

rpxdock.tests.motif.test_pairscore.**test_bin_score**(*respairscore*)

rpxdock.tests.motif.test_pairscore.**test_create_res_pair_score**(*respairdat, tmpdir*)

rpxdock.tests.motif.test_pairscore.**test_pair_score**(*respairscore*)

rpxdock.tests.motif.test_pairscore.**test_res_pair_score_pickle**(*respairscore, tmpdir*)

Module contents

rpxdock.tests.phmap package

Submodules

rpxdock.tests.phmap.test_phmap module

rpxdock.tests.phmap.test_phmap.**fa**(*a*)

rpxdock.tests.phmap.test_phmap.**test_phmap**()

rpxdock.tests.phmap.test_phmap.**test_phmap_contains**()

rpxdock.tests.phmap.test_phmap.**test_phmap_cpp_roundtrip**()

rpxdock.tests.phmap.test_phmap.**test_phmap_dump_load**(*tmpdir*)

rpxdock.tests.phmap.test_phmap.**test_phmap_eq**()

rpxdock.tests.phmap.test_phmap.**test_phmap_items**()

rpxdock.tests.phmap.test_phmap.**test_phmap_items_array**()

rpxdock.tests.phmap.test_phmap.**ua**(*a*)

Module contents

rpxdock.tests.rotamer package

Submodules

rpxdock.tests.rotamer.test_richardson module

rpxdock.tests.rotamer.test_richardson.**test_richardson_space**()

rpxdock.tests.rotamer.test_richardson.**test_richardson_space_cache**()

rpxdock.tests.rotamer.test_richardson.**test_richardson_space_concat**()

rpxdock.tests.rotamer.test_richardson.**test_richardson_space_nchi**()

Module contents

rpxdock.tests.sampling package

Submodules

rpxdock.tests.sampling.test_compound module

rpxdock.tests.sampling.test_compound.**test_compound_basic_indexing**()

rpxdock.tests.sampling.test_compound.**test_compound_get_xforms**()

rpxdock.tests.sampling.test_compound.**test_compound_ncell_dim**()

rpxdock.tests.sampling.test_compound.**test_compound_product_hier**()

rpxdock.tests.sampling.test_compound.**test_compound_single**()

rpxdock.tests.sampling.test_compound.**test_compound_split**()

rpxdock.tests.sampling.test_compound.**test_compound_split_cell**()

rpxdock.tests.sampling.test_compound.**test_compound_split_hier**()

rpxdock.tests.sampling.test_compound.**test_product_hier**()

rpxdock.tests.sampling.test_orientations module

rpxdock.tests.sampling.test_orientations.**test_karney_10**()

rpxdock.tests.sampling.test_orientations.**test_karney_by_covrad**()

rpxdock.tests.sampling.test_orientations.**test_karney_lengths_weights**()

rpxdock.tests.sampling.test_orientations.**test_karney_oddball**()

rpxdock.tests.sampling.test_orientations.**test_orientation_cpp**()

rpxdock.tests.sampling.test_orientations.**test_read_karney_orientations**()

rpxdock.tests.sampling.test_xform_grid module

rpxdock.tests.sampling.test_xform_grid.**test_grid_sym_axis**()

rpxdock.tests.sampling.test_xform_grid.**test_grid_sym_axis_flip**()

rpxdock.tests.sampling.test_xform_hierarchy module

rpxdock.tests.sampling.test_xform_hierarchy.**analyze_ori_hier**(*nside*, *resl*, *nsamp*)

rpxdock.tests.sampling.test_xform_hierarchy.**crappy_xform_hierarchy_resl_sanity_check**()

rpxdock.tests.sampling.test_xform_hierarchy.**slow_test_ori_hier_rand_nside4**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_OriCart1_hierarchy_product**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_RotCart1Hier_pickle**(*tmpdir*)

rpxdock.tests.sampling.test_xform_hierarchy.**test_accessors**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_avg_dist**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_cart_hier1**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_ori_hier_1cell**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_ori_hier_all2**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_ori_hier_angresl**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_ori_hier_rand**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_ori_hier_rand_nside**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_rot_hier**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_rot_hier_multicell**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_rotcart1_hier**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_rotcart1_hier_expand_top_N**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_rotcart1_hier_multicell**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_xform_hierarchy_ctor**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_xform_hierarchy_expand_top_N**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_xform_hierarchy_expand_top_N_nullval**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_xform_hierarchy_get_xforms**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_xform_hierarchy_get_xforms_bs**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_xform_hierarchy_plug_bug**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_xform_hierarchy_product**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_xform_hierarchy_product_zorder**()

rpxdock.tests.sampling.test_xform_hierarchy.**test_zorder**()

rpxdock.tests.sampling.test_xform_hierarchy.**urandint**(**args*)

rpxdock.tests.sampling.test_xform_hierarchy.**urange**(**args*)

Module contents

rpxdock.tests.search package

Submodules

rpxdock.tests.search.test_asym module

rpxdock.tests.search.test_asym.**main**()

rpxdock.tests.search.test_asym.**test_asym**(*hscore*, *body*, *body2*)

rpxdock.tests.search.test_asym.**test_asym_trim**(*hscore*, *body*, *body2*)

rpxdock.tests.search.test_asym.**testarg**()

rpxdock.tests.search.test_cyclic module

rpxdock.tests.search.test_cyclic.**get_arg**()

rpxdock.tests.search.test_cyclic.**test_make_cyclic_hier**(*hscore*, *body*)

rpxdock.tests.search.test_cyclic.**test_make_cyclic_hier_trim**(*hscore*, *body*)

rpxdock.tests.search.test_dockspec module

rpxdock.tests.search.test_dockspec.**test_dockspec**()

rpxdock.tests.search.test_gridslide module

rpxdock.tests.search.test_helix module

rpxdock.tests.search.test_helix.**main**()

rpxdock.tests.search.test_helix.**test_helix**(*hscore*, *body_tiny*)

rpxdock.tests.search.test_helix.**testarg**()

rpxdock.tests.search.test_hierarchical module

rpxdock.tests.search.test_multicomp module

rpxdock.tests.search.test_multicomp.**get_arg**(***kw*)

rpxdock.tests.search.test_multicomp.**test_cage_hier_3comp**(*hscore*, *bodyC4*, *bodyC3*, *bodyC2*)

rpxdock.tests.search.test_multicomp.**test_cage_hier_no_trim**(*hscore*, *body_cageA*, *body_cageB*)

rpxdock.tests.search.test_multicomp.**test_cage_hier_trim**(*hscore*, *body_cageA_extended*, *body_cageB_extended*)

rpxdock.tests.search.test_multicomp.**test_layer_hier_3comp**(*hscore*, *bodyC6*, *bodyC3*, *bodyC2*)

rpxdock.tests.search.test_onecomp module

rpxdock.tests.search.test_onecomp.**main**()

rpxdock.tests.search.test_onecomp.**test_cage_hier_D3_2_onecomp_notrim**(*hscore*, *bodyC2*)

rpxdock.tests.search.test_onecomp.**test_cage_hier_D3_onecomp_notrim**(*hscore*, *bodyC3*)

rpxdock.tests.search.test_onecomp.**test_cage_hier_onecomp_notrim**(*hscore*, *bodyC3*)

rpxdock.tests.search.test_plug module

rpxdock.tests.search.test_plug.**test_plug_hier**(*hscore*, *plug*, *hole*)

rpxdock.tests.search.test_plug.**test_plug_hier_trim**(*hscore*, *plug*, *hole*)

rpxdock.tests.search.test_plug.**test_plug_olig_grid**(*hscore*, *body_c3_mono*, *hole*)

rpxdock.tests.search.test_plug.**test_plug_olig_hier**(*hscore*, *body_c3_mono*, *hole*)

rpxdock.tests.search.test_plug.**testarg**()

rpxdock.tests.search.test_result module

rpxdock.tests.search.test_result.**test_mismatch_len**(*result*)

rpxdock.tests.search.test_result.**test_result**(*result*)

rpxdock.tests.search.test_result.**test_result_attrs**()

rpxdock.tests.search.test_result.**test_result_no_body_label**(*result*)

rpxdock.tests.search.test_result.**test_result_pickle**(*result*, *tmpdir*)

rpxdock.tests.search.test_result.**test_top_each**(*result*)

Module contents

rpxdock.tests.util package

Submodules

rpxdock.tests.util.test_bunch module

rpxdock.tests.util.test_bunch.**test_bunch_init**()

rpxdock.tests.util.test_bunch.**test_bunch_items**()

rpxdock.tests.util.test_bunch.**test_bunch_pickle**(*tmpdir*)

rpxdock.tests.util.test_bunch.**test_bunch_sub**()

rpxdock.tests.util.test_cache module

rpxdock.tests.util.test_cache.**sum_**(**args*)

rpxdock.tests.util.test_cache.**test_cache**()

rpxdock.tests.util.test_numeric module

rpxdock.tests.util.test_numeric.**test_eig_svd**()

rpxdock.tests.util.test_numeric.**test_pca**()

rpxdock.tests.util.test_pybind_types module

rpxdock.tests.util.test_pybind_types.**test_xform_round_trip**()

rpxdock.tests.util.test_pybind_types.**test_xform_round_trip_2d**()

rpxdock.tests.util.test_timer module

rpxdock.tests.util.test_timer.**test_summary**()

rpxdock.tests.util.test_timer.**test_timer**()

rpxdock.tests.util.test_util module

class rpxdock.tests.util.test_util.**dummyclass**
Bases: object

rpxdock.tests.util.test_util.**dummyfunc**()

rpxdock.tests.util.test_util.**test_can_pickle**()

rpxdock.tests.util.test_util.**test_dilated_int**()

rpxdock.tests.util.test_util.**test_load_threads**(*tmpdir*)

rpxdock.tests.util.test_util.**test_num_digits**()

rpxdock.tests.util.test_util.**test_sanitize_for_pickle**()

Module contents

rpxdock.tests.xbin package

Submodules

rpxdock.tests.xbin.test_smear module

rpxdock.tests.xbin.test_smear.**check_scores**(*s0*, *s1*)

rpxdock.tests.xbin.test_smear.**smear_bench**()

rpxdock.tests.xbin.test_smear.**test_smear_multiple**()

rpxdock.tests.xbin.test_smear.**test_smear_one**()

rpxdock.tests.xbin.test_smear.**test_smear_one_bounding**()

rpxdock.tests.xbin.test_smear.**test_smear_one_exhalf_oddori_sphere**()

rpxdock.tests.xbin.test_smear.**test_smear_one_kernel**()

rpxdock.tests.xbin.test_smear.**test_smear_one_oddori**()

rpxdock.tests.xbin.test_smear.**test_smear_one_oddori_sphere**()

rpxdock.tests.xbin.test_smear.**test_smear_two**()

rpxdock.tests.xbin.test_xbin module

rpxdock.tests.xbin.test_xbin.**test_create_binner**()

rpxdock.tests.xbin.test_xbin.**test_key_of**()

rpxdock.tests.xbin.test_xbin.**test_pickle**(*tmpdir*)

rpxdock.tests.xbin.test_xbin.**test_xbin_covrad**(*niter=20*, *nsamp=5000*)

rpxdock.tests.xbin.test_xbin.**test_xbin_covrad_ori**()

rpxdock.tests.xbin.test_xbin.**test_xbin_cpp**()

rpxdock.tests.xbin.test_xbin_util module

rpxdock.tests.xbin.test_xbin_util.**test_key_of_pairs**()

rpxdock.tests.xbin.test_xbin_util.**test_map_of_selected_pairs**()

rpxdock.tests.xbin.test_xbin_util.**test_selected_pairs_pos**()

rpxdock.tests.xbin.test_xbin_util.**test_sskey_of_selected_pairs**()

rpxdock.tests.xbin.test_xbin_util.**test_ssmap_of_selected_pairs**()

Module contents

Submodules

rpxdock.tests.conftest module

rpxdock.tests.test_homog module

rpxdock.tests.test_homog.**test_align_around_axis**()

rpxdock.tests.test_homog.**test_align_lines_dof_dihedral_basic**()

rpxdock.tests.test_homog.**test_align_lines_dof_dihedral_rand**(*n=100*)

rpxdock.tests.test_homog.**test_align_lines_dof_dihedral_rand_3D**()

rpxdock.tests.test_homog.**test_align_lines_dof_dihedral_rand_single**()

rpxdock.tests.test_homog.**test_align_vectors_minangle**()

rpxdock.tests.test_homog.**test_align_vectors_una_case**()

rpxdock.tests.test_homog.**test_angle**()

rpxdock.tests.test_homog.**test_axis_ang_cen_of_rand**()

rpxdock.tests.test_homog.**test_axis_angle_of**()

rpxdock.tests.test_homog.**test_axis_angle_of_rand**()

rpxdock.tests.test_homog.**test_calc_dihedral_angle**()

rpxdock.tests.test_homog.**test_dihedral**()

rpxdock.tests.test_homog.**test_expand_xforms_basic**()

rpxdock.tests.test_homog.**test_hcross**()

rpxdock.tests.test_homog.**test_hinv_rand**()

rpxdock.tests.test_homog.**test_homo_rotation_angle**()

rpxdock.tests.test_homog.**test_homo_rotation_array**()

rpxdock.tests.test_homog.**test_homo_rotation_center**()

rpxdock.tests.test_homog.**test_homo_rotation_single**()

rpxdock.tests.test_homog.**test_hstub**()

rpxdock.tests.test_homog.**test_htrans**()

rpxdock.tests.test_homog.**test_intersect_planes**()

rpxdock.tests.test_homog.**test_intersect_planes_rand**()

rpxdock.tests.test_homog.**test_is_valid_rays**()

rpxdock.tests.test_homog.**test_line_line_closest_points**()

rpxdock.tests.test_homog.**test_line_line_dist**()

rpxdock.tests.test_homog.**test_place_lines_to_isect_F432**()

rpxdock.tests.test_homog.**test_place_lines_to_isect_F432_null**()

rpxdock.tests.test_homog.**test_place_lines_to_isect_onecase**()

rpxdock.tests.test_homog.**test_point_in_plane**()

rpxdock.tests.test_homog.**test_proj_prep**()

rpxdock.tests.test_homog.**test_rand_ray**()

rpxdock.tests.test_homog.**test_ray_in_plane**()

rpxdock.tests.test_homog.**test_sym**()

Module contents

rpxdock.util package

Submodules

rpxdock.util.bunch module

class rpxdock.util.bunch.**Bunch**(*_Bunch__arg_or_ns=None*, ***kw*)
Bases: dict

copy() → a shallow copy of D

static from_dict(*d*)

sub(*_Bunch__BUNCH_SUB_ITEMS=None*, ***kw*)

toDict()

rpxdock.util.bunch.**bunchify**(*x*)

rpxdock.util.bunch.**unbunchify**(*x*)

rpxdock.util.cache module

class rpxdock.util.cache.**Cache**
Bases: dict

checkpoint()

get_cached(*fun*, **args*, *_force_reload=False*, *_saved_only=False*, *_nodump=False*, *_key=None*, ***kw*)

key_of(*fun*, **args*, ***kw*)

keys_have_changed_since_checkpoint()

load(*fname*, *strict=True*)

remove(*fun*, **args*, *_force_reload=False*, ***kw*)

save(*fname*, *force=False*)

class rpxdock.util.cache.**CachedProxy**(*thing*)
Bases: object

rpxdock.util.cache.**NOCACHE**(*fun*, **args*, ***kw*)

rpxdock.util.cache.**remove_proxy**(*thing*)

rpxdock.util.dilated_int_test module

rpxdock.util.dilated_int_test.**TEST_dilated_int_64bit**() → bool

rpxdock.util.numeric module

rpxdock.util.numeric.**pca**(*coords*)

rpxdock.util.numeric.**pca_eig**(*coords*)

rpxdock.util.numeric.**svd**(*x*)

rpxdock.util.parallel_build_modules module

rpxdock.util.parallel_build_modules.**check_needs_update**(*filepath*)

rpxdock.util.parallel_build_modules.**files_needing_rebuild**()

rpxdock.util.parallel_build_modules.**fullname_from_path**(*f*)

rpxdock.util.parallel_build_modules.**maybe_build**(*f*)

rpxdock.util.parallel_build_modules.**parallel_build_modules**(*cppfiles=None*)

rpxdock.util.plot module

rpxdock.util.plot.**get_plotter**(**args*, ***kw*)

rpxdock.util.plot.**hist**(**args*, *title=''*, *show=True*, ***kw*)

rpxdock.util.plot.**scatter**(**args*, *title=''*, *show=True*, *xscale='linear'*, *lines=False*, *figsize=16, 9*, ***kw*)

rpxdock.util.plot.**show**()

rpxdock.util.plot.**subplots**(*x*, *y*, *figsize=None*, *rowmajor=False*, ***kw*)

rpxdock.util.pybind_types_test module

rpxdock.util.pybind_types_test.**test_xform_round_trip**(*arg0: numpy.ndarray[float64]*) → numpy.ndarray[float64]

rpxdock.util.timer module

class rpxdock.util.timer.**Timer**(*name='Timer'*, *verbose=False*)

Bases: object

alltimes(*name*)

checkpoint(*name='none'*, *verbose=False*)

merge(*others*)

report(*order='longest'*, *summary='sum'*, *namelen=None*, *precision='10.5f'*, *printme=True*)

report_dict (*order='longest'*, *summary='sum'*)

start ()

stop ()

property total

rpxdock.util.util module

class rpxdock.util.util.**InProcessExecutor** (**args*, ***kw*)
Bases: object

submit (*fn*, **args*, ***kw*)

class rpxdock.util.util.**NonFuture** (*fn*, **args*, *dummy=None*, ***kw*)
Bases: object

result ()

rpxdock.util.util.**can_pickle** (*thing*)

rpxdock.util.util.**cpu_count** ()

rpxdock.util.util.**dump** (*thing*, *f*)

rpxdock.util.util.**dump_str** (*string*, *f*)

rpxdock.util.util.**hash_str_to_int** (*s*)

rpxdock.util.util.**load** (*f*, *verbose=True*)

rpxdock.util.util.**load_threads** (*fnames*, *nthread=0*)

rpxdock.util.util.**num_digits** (*n*)

rpxdock.util.util.**pickle_analysis** (*thing*, *mintime=0.1*, *loglevel='debug'*)

rpxdock.util.util.**pickle_time** (*thing*)

rpxdock.util.util.**sanitize_for_pickle** (*data*)

Module contents

rpxdock.xbin package

Submodules

rpxdock.xbin.smear module

rpxdock.xbin.smear.**smear** (**args*, ***kwargs*)
Overloaded function.

1. smear(xbin: rpxdock.xbin.xbin.Xbin_double, phmap: rpxdock::phmap::PHMap<unsigned long, double>, radius: int = 1, extrahalf: bool = False, oddlast3: bool = True, sphere: bool = True, kernel: numpy.ndarray[float64[m, 1]] = array([], dtype=float64)) -> rpxdock::phmap::PHMap<unsigned long, double>

smear out xmap into neighbor cells

2. smear(xbin: rpxdock.xbin.xbin.Xbin_float, phmap: rpxdock::phmap::PHMap<unsigned long, double>, radius: int = 1, extrahalf: bool = False, oddlast3: bool = True, sphere: bool = True, kernel: numpy.ndarray[float64[m, 1]] = array([], dtype=float64)) -> rpxdock::phmap::PHMap<unsigned long, double>

smear out xmap into neighbor cells

3. smear(xbin: rpxdock.xbin.xbin.Xbin_double, phmap: rpxdock::phmap::PHMap<unsigned long, float>, radius: int = 1, extrahalf: bool = False, oddlast3: bool = True, sphere: bool = True, kernel: numpy.ndarray[float32[m, 1]] = array([], dtype=float32)) -> rpxdock::phmap::PHMap<unsigned long, float>

smear out xmap into neighbor cells

4. smear(xbin: rpxdock.xbin.xbin.Xbin_float, phmap: rpxdock::phmap::PHMap<unsigned long, float>, radius: int = 1, extrahalf: bool = False, oddlast3: bool = True, sphere: bool = True, kernel: numpy.ndarray[float32[m, 1]] = array([], dtype=float32)) -> rpxdock::phmap::PHMap<unsigned long, float>

smear out xmap into neighbor cells

rpxdock.xbin.xbin module

class `rpxdock.xbin.xbin.`**`Xbin_double`**

Bases: `pybind11_builtins.pybind11_object`

`F6_to_xform`(*self: rpxdock.xbin.xbin.Xbin_double, arg0: numpy.ndarray[float64[m, n]], arg1: numpy.ndarray[uint64[m, 1]]*) → numpy.ndarray[float64]

`bincen_of`(*self: rpxdock.xbin.xbin.Xbin_double, arg0: numpy.ndarray[uint64[m, 1], flags.writeable]*) → numpy.ndarray[float64]

property cart_resl

property grid6

`key_of`(*self: rpxdock.xbin.xbin.Xbin_double, xform: numpy.ndarray[float64]*) → numpy.ndarray[uint64[m, 1]]

key of xform

property max_cart

`ori_cell_of`(*self: rpxdock.xbin.xbin.Xbin_double, xform: numpy.ndarray[float64]*) → numpy.ndarray[uint64[m, 1]]

key of xform

property ori_nside

property ori_resl

`xform_to_F6`(*self: rpxdock.xbin.xbin.Xbin_double, arg0: numpy.ndarray[float64]*) → tuple

class `rpxdock.xbin.xbin.`**`Xbin_float`**

Bases: `pybind11_builtins.pybind11_object`

`F6_to_xform`(*self: rpxdock.xbin.xbin.Xbin_float, arg0: numpy.ndarray[float32[m, n]], arg1: numpy.ndarray[uint64[m, 1]]*) → numpy.ndarray[float32]

`bincen_of`(*self: rpxdock.xbin.xbin.Xbin_float, arg0: numpy.ndarray[uint64[m, 1], flags.writeable]*) → numpy.ndarray[float32]

property cart_resl

property grid6

key_of (*self:* rpxdock.xbin.xbin.Xbin_float, *xform:* *numpy.ndarray[float32]*) → numpy.ndarray[uint64[m, 1]]
key of xform

property max_cart

ori_cell_of (*self:* rpxdock.xbin.xbin.Xbin_float, *xform:* *numpy.ndarray[float32]*) → numpy.ndarray[uint64[m, 1]]
key of xform

property ori_nside

property ori_resl

xform_to_F6 (*self:* rpxdock.xbin.xbin.Xbin_float, *arg0: numpy.ndarray[float32]*) → tuple

rpxdock.xbin.xbin.**create_Xbin_nside_double** (*arg0: float*, *arg1: int*, *arg2: float*) → *rpxdock.xbin.xbin.Xbin_double*

rpxdock.xbin.xbin.**create_Xbin_nside_float** (*arg0: float*, *arg1: int*, *arg2: float*) → *rpxdock.xbin.xbin.Xbin_float*

rpxdock.xbin.xbin_test module

rpxdock.xbin.xbin_test.**TEST_XformHash_XformHash_bt24_BCC6** () → bool

rpxdock.xbin.xbin_util module

rpxdock.xbin.xbin_util.**key_of_pairs** (**args*, ***kwargs*)
Overloaded function.

1. **key_of_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, pairs: array, xform1: array, xform2: array, pos1: numpy.ndarr**
[0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[uint64[m, 1]]

2. **key_of_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, pairs: array, xform1: array, xform2: array, pos1: numpy.ndarray**
[0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[uint64[m, 1]]

rpxdock.xbin.xbin_util.**key_of_selected_pairs** (**args*, ***kwargs*)
Overloaded function.

1. **key_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, idx1: array, idx2: array, xform1: array, xform2: array**
[0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[uint64[m, 1]]

2. **key_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, idx: array, xform1: array, xform2: array, pos1: numpy**
[0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[uint64[m, 1]]

3. **key_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, idx1: array, idx2: array, xform1: array, xform2: array, p**
[0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[uint64[m, 1]]

4. **key_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, idx: array, xform1: array, xform2: array, pos1: numpy.n**
[0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[uint64[m, 1]]

`rpxdock.xbin.xbin_util.`**`map_of_selected_pairs`**(**args*, ***kwargs*)

Overloaded function.

1. **map_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, phmap: rpxdock::phmap::PHMap<unsigned long, do**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[float64[m, 1]]

2. **map_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, phmap: rpxdock::phmap::PHMap<unsigned long, do**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[float64[m, 1]]

3. **map_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, phmap: rpxdock::phmap::PHMap<unsigned long, doub**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[float64[m, 1]]

4. **map_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, phmap: rpxdock::phmap::PHMap<unsigned long, doub**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[float64[m, 1]]

`rpxdock.xbin.xbin_util.`**`map_pairs_multipos`**(**args*, ***kwargs*)

Overloaded function.

1. **map_pairs_multipos(xbin: rpxdock.xbin.xbin.Xbin_double, phmap: rpxdock::phmap::PHMap<unsigned long, doub**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[float64[m, 1]]

2. **map_pairs_multipos(xbin: rpxdock.xbin.xbin.Xbin_float, phmap: rpxdock::phmap::PHMap<unsigned long, double**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[float64[m, 1]]

`rpxdock.xbin.xbin_util.`**`sskey_of_selected_pairs`**(**args*, ***kwargs*)

Overloaded function.

1. **sskey_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, idx1: array, idx2: array, ss1: array, ss2: array, xform**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[uint64[m, 1]]

2. **sskey_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, idx: array, ss1: array, ss2: array, xform1: array, xfor**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[uint64[m, 1]]

3. **sskey_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, idx1: array, idx2: array, ss: array, xform: array, pos**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[uint64[m, 1]]

4. **sskey_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, idx: array, ss: array, xform: array, pos1: numpy.nda**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[uint64[m, 1]]

5. **sskey_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, idx1: array, idx2: array, ss1: array, ss2: array, xform1:**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[uint64[m, 1]]

6. **sskey_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, idx: array, ss1: array, ss2: array, xform1: array, xform**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[uint64[m, 1]]

7. **sskey_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, idx1: array, idx2: array, ss: array, xform: array, pos1:**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[uint64[m, 1]]

8. **sskey_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, idx: array, ss: array, xform: array, pos1: numpy.ndarra**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[uint64[m, 1]]

`rpxdock.xbin.xbin_util.`**`ssmap_of_selected_pairs`**(**args*, ***kwargs*)

Overloaded function.

1. **ssmap_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, phmap: rpxdock::phmap::PHMap<unsigned long, d**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[float64[m, 1]]

2. **ssmap_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_double, phmap: rpxdock::phmap::PHMap<unsigned long, d**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[float64[m, 1]]

3. **ssmap_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, phmap: rpxdock::phmap::PHMap<unsigned long, dou**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[float64[m, 1]]

4. **ssmap_of_selected_pairs(xbin: rpxdock.xbin.xbin.Xbin_float, phmap: rpxdock::phmap::PHMap<unsigned long, dou**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32[4, 4]] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[float64[m, 1]]

`rpxdock.xbin.xbin_util.`**`ssmap_pairs_multipos`**(**args*, ***kwargs*)

Overloaded function.

1. **ssmap_pairs_multipos(xbin: rpxdock.xbin.xbin.Xbin_double, phmap: rpxdock::phmap::PHMap<unsigned long, do**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]]), pos2: numpy.ndarray[float64] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]])) -> numpy.ndarray[float64[m, 1]]

2. **ssmap_pairs_multipos(xbin: rpxdock.xbin.xbin.Xbin_float, phmap: rpxdock::phmap::PHMap<unsigned long, doub**
   [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32), pos2: numpy.ndarray[float32] = array([[1., 0., 0., 0.], [0., 1., 0., 0.], [0., 0., 1., 0.], [0., 0., 0., 1.]], dtype=float32)) -> numpy.ndarray[float64[m, 1]]

Module contents

7.1.2 Submodules

7.1.3 rpxdock.homog module

`rpxdock.homog.`**`align_around_axis`**(*axis*, *u*, *v*)

`rpxdock.homog.`**`align_lines_isect_axis2`**(*pt1*, *ax1*, *pt2*, *ax2*, *ta1*, *tp1*, *ta2*, *sl2*)

`rpxdock.homog.`**`align_vector`**(*a*, *b*)

`rpxdock.homog.`**`align_vectors`**(*a1*, *a2*, *b1*, *b2*)

minimizes angular error

`rpxdock.homog.`**`angle`**(*u*, *v*)

rpxdock.homog.**angle_degrees**(*u*, *v*)

rpxdock.homog.**angle_of**(*xforms*)

rpxdock.homog.**angle_of_3x3**(*rots*)

rpxdock.homog.**axis_ang_cen_of**(*xforms*, *debug=False*)

rpxdock.homog.**axis_ang_cen_of_eig**(*xforms*, *debug=False*)

rpxdock.homog.**axis_ang_cen_of_planes**(*xforms*, *debug=False*)

rpxdock.homog.**axis_angle_of**(*xforms*)

rpxdock.homog.**axis_angle_of_3x3**(*rots*)

rpxdock.homog.**calc_dihedral_angle**(*p1*, *p2*, *p3*, *p4*)

rpxdock.homog.**dihedral**(*p1*, *p2*, *p3*, *p4*)

rpxdock.homog.**expand_xforms**(*G*, *N=3*, *redundant_point=array([1.0, 3.0, 10.0, 1.0])*, *maxrad=9000000000.0*)

rpxdock.homog.**fast_axis_of**(*xforms*)

rpxdock.homog.**guess_is_degrees**(*angle*)

rpxdock.homog.**h_rand_points**(*shape=1*)

rpxdock.homog.**hcross**(*a*, *b*)

rpxdock.homog.**hdot**(*a*, *b*)

rpxdock.homog.**hinv**(*xforms*)

rpxdock.homog.**hnorm**(*a*)

rpxdock.homog.**hnorm2**(*a*)

rpxdock.homog.**hnormalized**(*a*)

rpxdock.homog.**hpoint**(*point*)

rpxdock.homog.**hray**(*origin*, *direction*)

rpxdock.homog.**hrot**(*axis*, *angle*, *center=None*, *dtype='f8'*, ***args*)

rpxdock.homog.**hstub**(*u*, *v*, *w*, *cen=None*)

rpxdock.homog.**htrans**(*trans*, *dtype='f8'*)

rpxdock.homog.**hvec**(*vec*)

rpxdock.homog.**intersect_planes**(*plane1*, *plane2*)

intersect_Planes: find the 3D intersection of two planes Input: two planes represented by rays shape=(. . . , 4, 2) Output: L = the intersection line (when it exists) Return: rays shape=(. . . ,4,2), status

0 = intersection returned 1 = disjoint (no intersection) 2 = the two planes coincide

rpxdock.homog.**is_broadcastable**(*shp1*, *shp2*)

rpxdock.homog.**is_homog_xform**(*xforms*)

rpxdock.homog.**is_valid_quat_rot**(*quat*)

rpxdock.homog.**is_valid_rays**(*r*)

rpxdock.homog.**line_angle**(*u*, *v*)

rpxdock.homog.**line_angle_degrees**(*u*, *v*)

rpxdock.homog.**line_line_closest_points**(*ray1*, *ray2*, *verbose=0*)
currently errors if ax1==ax2

rpxdock.homog.**line_line_closest_points_pa**(*pt1*, *ax1*, *pt2*, *ax2*, *verbose=0*)

rpxdock.homog.**line_line_distance**(*ray1*, *ray2*)

rpxdock.homog.**line_line_distance_pa**(*pt1*, *ax1*, *pt2*, *ax2*)

rpxdock.homog.**point_in_plane**(*plane*, *pt*)

rpxdock.homog.**proj_perp**(*u*, *v*)

rpxdock.homog.**quat_multiply**(*q*, *r*)

rpxdock.homog.**quat_to_rot**(*quat*, *dtype='f8'*, *shape=3, 3*)

rpxdock.homog.**quat_to_upper_half**(*quat*)

rpxdock.homog.**quat_to_xform**(*quat*, *dtype='f8'*)

rpxdock.homog.**rand_point**(*shape=()*)

rpxdock.homog.**rand_quat**(*shape=()*)

rpxdock.homog.**rand_ray**(*shape=()*, *cen=0, 0, 0*, *sdev=1*)

rpxdock.homog.**rand_unit**(*shape=()*)

rpxdock.homog.**rand_vec**(*shape=()*)

rpxdock.homog.**rand_xform**(*shape=()*, *cart_cen=0*, *cart_sd=1*)

rpxdock.homog.**rand_xform_aac**(*shape=()*, *axis=None*, *ang=None*, *cen=None*)

rpxdock.homog.**ray_in_plane**(*plane*, *ray*)

rpxdock.homog.**rot**(*axis*, *angle*, *degrees='auto'*, *dtype='f8'*, *shape=3, 3*)

rpxdock.homog.**rot_to_quat**(*xform*)

rpxdock.homog.**rotation_around_dof_for_target_angle**(*target_angle*, *dof_angle*, *fix_to_dof_angle*)

rpxdock.homog.**xform_around_dof_for_vector_target_angle**(*fix*, *mov*, *dof*, *target_angle*)

rpxdock.homog.**xform_to_quat**(*xform*)

7.1.4 Module contents

CHAPTER

EIGHT

SUMMARY

Rpxdock (as well as the ambitious vaporware scheme, and the conceptually related rifdock) is a multi-scale model of protein structure suited to global search of conformation space. rpxdock utilizes a novel transform-based objective function which *retains some of the power of fullatom force-fields*, while avoiding a costly and difficult-to-optimize fullatom model. The rpxdock model is carefully crafted to allow both *pair* and *hierarchical* decomposition of all underlying DOFs, opening the door to new optimization techniques like *Hierarchical Sampling and Scoring* and *Hierarchical Packing*. Rpxdock and related projects are currently in use in the baker lab, and *seem to perform well*.

CHAPTER

NINE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

INDEX

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z