VoroIF: analysis of interfaces in protein complexes using Voronoi tessellations and graph neural networks

Kliment Olechnovič

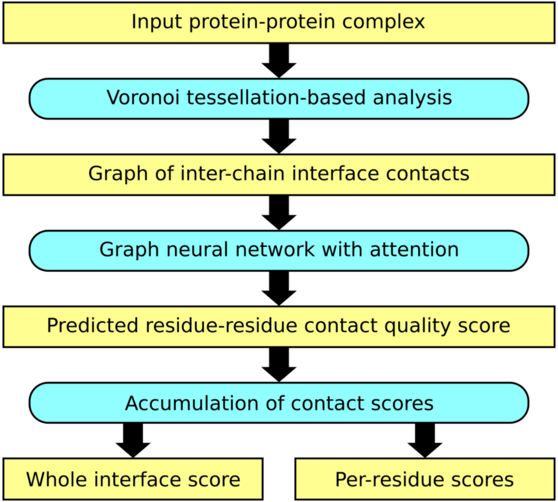
Vilnius University / Life Sciences Center / Institute of Biotechnology

December 2022

Life Sciences Center

Institute of Biotechnology

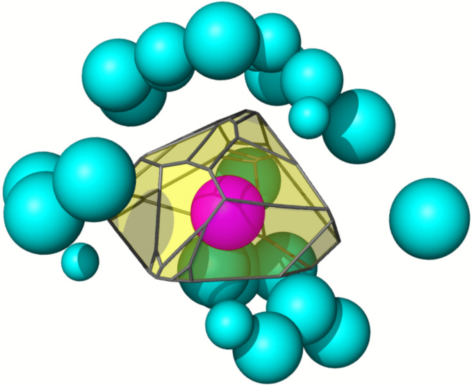
General scheme of VoroIF-GNN



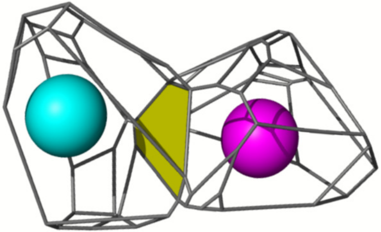
Voronoi tessellation-derived interface representation

Deriving atom-atom contacts

Voronoi cell of an atom surrounded by its neighbors

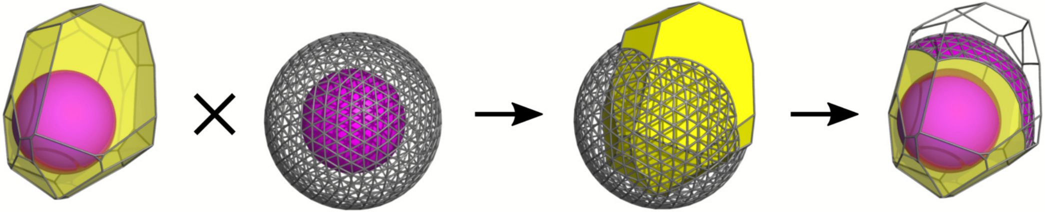


Atom-atom contact surface defined as the face shared by two adjacent Voronoi cells.



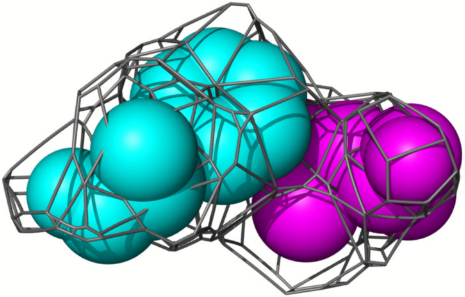
Constraining CSAs and deriving SASA for an atom

- **CSA** — contact surface area
- **SASA** — surface-accessible surface area

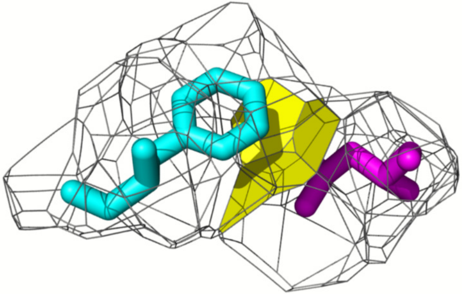


Deriving residue-residue contacts

Voronoi cells of two neighboring residues

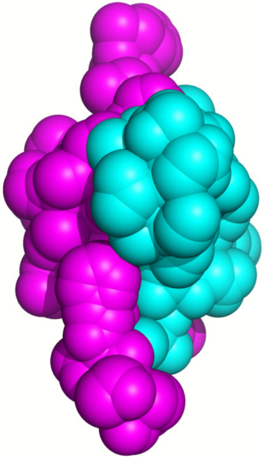


Residue-residue contact surface defined as a union of atom-atom contact surfaces

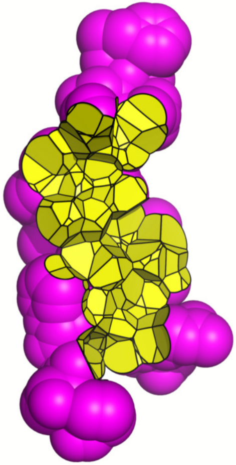


SASAs and intersubunit interfaces

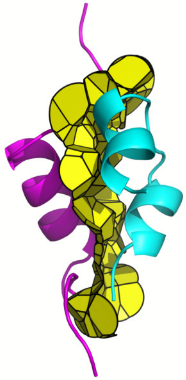
Solvent-accessible surface of an insulin heterodimer PDB:4UNG colored by subunit



The intersubunit interface shown together with the SAS of one subunit



The intersubunit interface shown together with both subunits represented as cartoons



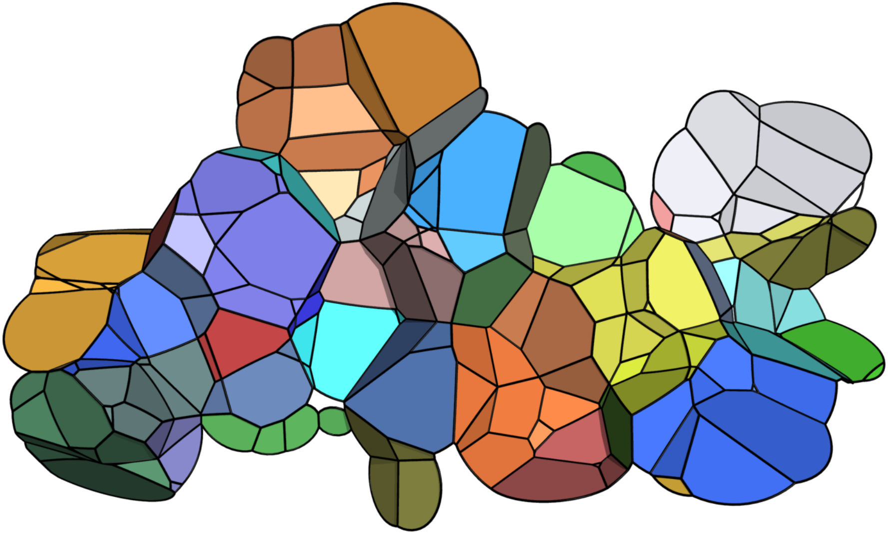
Interface graph definition

Important note about the interface graphs

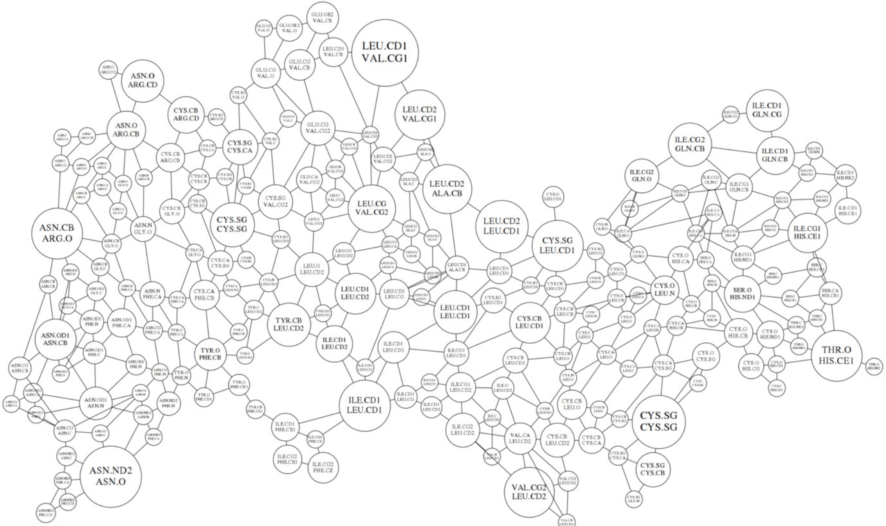
Our interface graphs are fairly unusual:

- **Graph nodes** correspond to inter-chain contacts (on atom-atom or residue-residue levels)
- **Graph edges** correspond to borders between adjacent atom-atom or residue-residue contacts

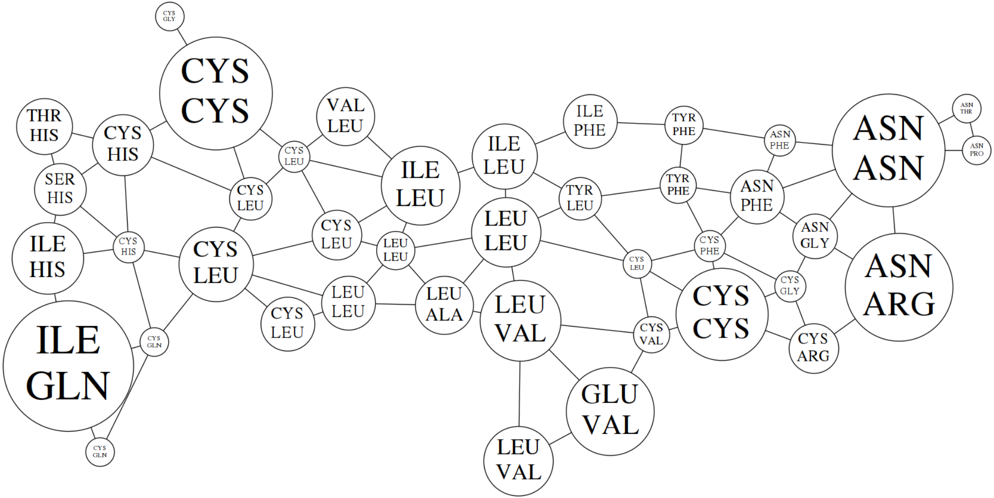
Interface graph example — source



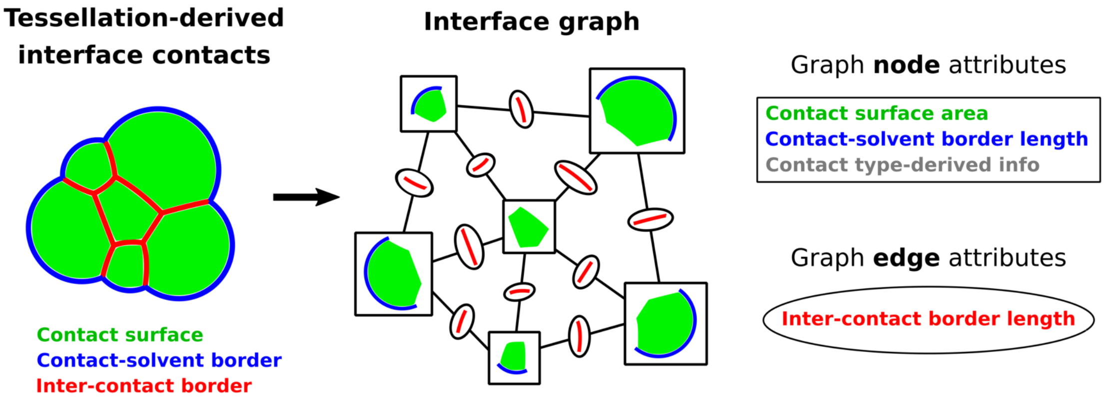
Interface graph example — inter-atom level



Interface graph example — inter-residue level



Interface graph construction and annotation



Contact-type derived info in nodes

A node representing a contact between two atoms of types A and B was annotated using the type-dependent coefficients from our contact area-based statistical potential VoroMQA:

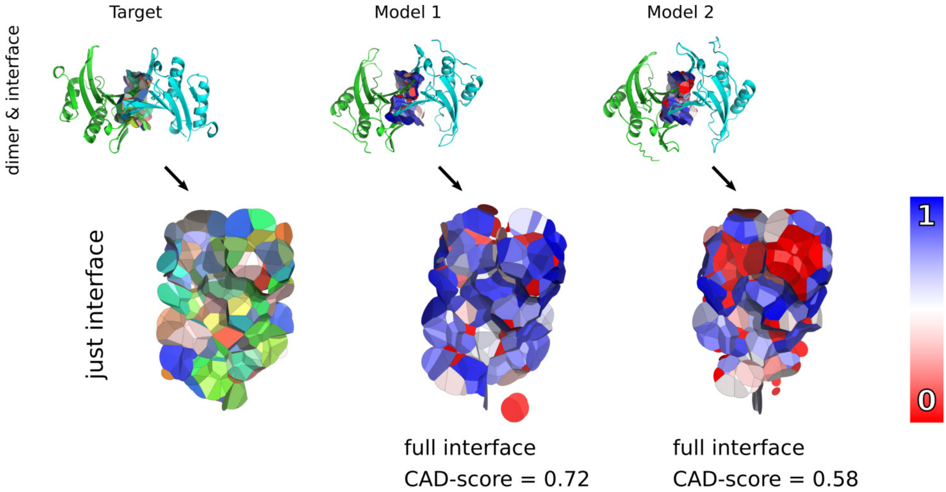
- $\mathrm{VoroMQA}_{\mathrm{coef}}(A, B) * \mathrm{area}$
- $\mathrm{VoroMQA}_{\mathrm{coef}}(A, \mathit{solvent}) * \mathrm{area}$
- $\mathrm{VoroMQA}_{\mathrm{coef}}(B, \mathit{solvent}) * \mathrm{area}$

When going from atom-level to residue-level nodes, the VoroMQA-based values were simply summed.

What to predict for an interface graph

What values to predict for graph nodes

Ground truth values for graph nodes are derived from CAD-score (Contact Area Difference score) values of residue-residue contacts.

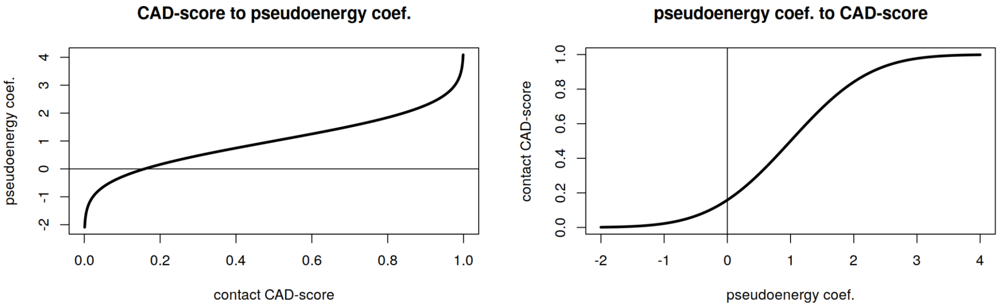


Pseudoenergy trick

Node level scores must behave like a ”pseudoenergy“:

- must be "summable", so that global or residue score = the sum of node scores
- must be weighted by corresponding contact areas
- very bad scores must penalize the total sum

```
pseudoenergy = (qnorm(cad_score)+shift)*area      # shift=1 was the best
cad_score = pnorm(pseudoenergy/area-shift)
```



Data for machine learning

Generating datasets

Training/testing/validation sets were constructed as follows:

- a non-redundant set of 1567 heterodimers were downloaded from PDB using the clustering information provided by PPI3D
- the whole set was split into three sets: training/validation/testing containing 1097/235/235 heterodimers
- for each native structure (target), redocking was performed with FTDock, CAD-score values were computed and a nonredundant set of models of varying quality was selected (usually about 15-20 models for a target)
- each per-target set included models with at least partially correct binding site, but completely wrong interface (this made the model scoring and selection tasks **substantially difficult**)

Example of a set of docking models

ID	x	y	z	a1	a2	a3	cadscore	site_cadscore
1E50_2250	-7	27	4	45	153	90	0.74375	0.87635
1E50_32	-13	25	2	18	153	90	0.63728	0.75543
1E50_2735	-7	28	1	72	162	120	0.53173	0.68644
1E50_15946	-16	26	-2	45	162	120	0.38075	0.55364
1E50_10393	-16	28	5	0	153	90	0.24134	0.47034
1E50_3759	7	29	7	351	117	40	0.13939	0.51889
1E50_17192	24	22	8	315	63	0	0.0386	0.42122
1E50_15006	-13	27	13	342	18	0	0	0.40432
1E50_5533	28	-13	20	0	45	204	0	0.30295
1E50_14280	27	-22	-22	180	126	60	0	0.20266
1E50_532	34	4	-18	207	54	100	0	0.10126
1E50_20368	1	-39	10	324	117	80	0	0.00119
1E50_9297	37	5	-22	261	54	80	0	0

Applying graph neural networks

GNN architecture selection and application

Initial ideas for the graph neural network (GNN):

- train to predict node scores (i.e. train to minimize MSE loss between predicted and ground truth CAD-score pseudoenergies)
- use both node and edge features in an attention mechanism
- in the validation stage, judge GNN performance by assessing how a global score (equal to the sum of node predictions) is able to select the best multimeric model out of many

A multilayer GNN based on on GATv2 convolutional operator was chosen, because in GATv2 the edge features are used straightforwardly when computing attention coefficients:

$$\alpha_{i,j} = \frac{\exp\left(\mathbf{a}^{\top}\mathrm{LeakyReLU}(\hat{}[\mathbf{x}_i \parallel \mathbf{x}_j \parallel \mathbf{e}_{i,j}])\right)}{\sum_{k\in\mathcal{N}(i)\cup\{i\}} \exp(\mathbf{a}^{\top}\mathrm{LeakyReLU}(\hat{}[\mathbf{x}_i \parallel \mathbf{x}_k \parallel \mathbf{e}_{i,k}]))}$$

Selected GNN architecture hyperparameters

ML experiments resulted in selecting a 4-layer GATv2 architecture with 8 attention heads per layer:

```
class GNN(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1=torch_geometric.nn.GATv2Conv(15, 16, heads=8, edge_dim=1, add_self_loops=False, dropout=0.25)
        self.conv2=torch_geometric.nn.GATv2Conv(16*8, 16, heads=8, edge_dim=1, add_self_loops=False, dropout=0.25)
        self.conv3=torch_geometric.nn.GATv2Conv(16*8, 16, heads=8, edge_dim=1, add_self_loops=False, dropout=0.25)
        self.conv4=torch_geometric.nn.GATv2Conv(16*8, 8, heads=8, edge_dim=1, add_self_loops=False, dropout=0.25)
        self.lin1=torch.nn.Linear(8*8, 1)

    def forward(self, data):
        x=data.x
        x=self.conv1(x, data.edge_index, data.edge_attr)
        x=torch.nn.functional.elu(x)
        x=self.conv2(x, data.edge_index, data.edge_attr)
        x=torch.nn.functional.elu(x)
        x=self.conv3(x, data.edge_index, data.edge_attr)
        x=torch.nn.functional.elu(x)
        x=self.conv4(x, data.edge_index, data.edge_attr)
        return self.lin1(x)
```

Final testing results

Performance of the final method on a 235 sets of dimeric models generated by redocking and not used in training:

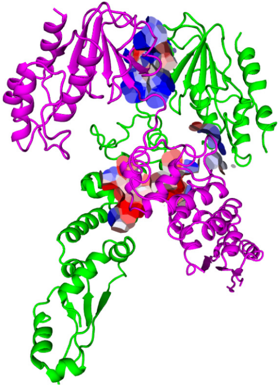
Selection method	**Rate of correct top 1**	**Mean interface CAD-score**	**Mean z-score of interface CAD-score**
Ideal	100%	0.78	1.85
VoroIF-GNN (new)	86%	0.74	1.72
VoroMQA energy (old)	53%	0.63	1.34

Case study of T1121 (PDB 7til)

Model
T1121TS205_3o

pCAD-score = 0.87

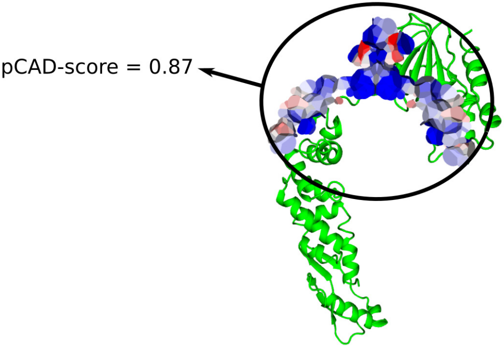
Target
PDB 7til



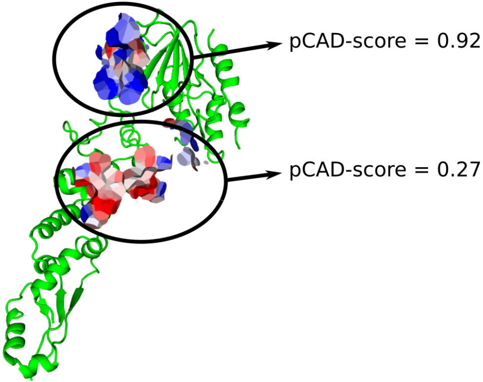
pCAD-score = 0.68

Case study of T1121 (PDB 7til)

Model
T1121TS205_3o



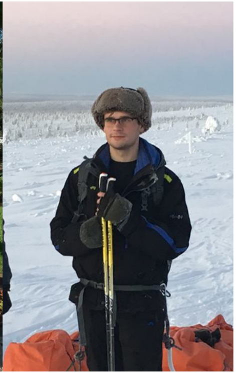
Target
PDB 7til



Conclusions

- VoroIF is very local
- VoroIF is relatively good at scoring interfaces, but not really suited for per-residue scores that were required by CASP15
- VoroIF is unusual, but it works - so it may be especially useful when combined with other scoring methods

Acknowledgments



Life Sciences Center

Institute of Biotechnology

Funding: Research Council of Lithuania

CASP15 Team

Justas Dapkūnas

Lukas Valančauskas

Česlovas Venclovas

bioinformatics.lt

Rytis Dičiūnas

Kęstutis Timinskas

Albertas Timinskas

Darius Kazlauskas

Mindaugas Margelevičius

Visvaldas Kairys