基于Voronoi分解方法的晶体结构分析库（CAVD）

V1.0

源程序

叶安江

20180816

# CAVD代码说明

CAVD使用底层的C++代码实现计算，使用Cython语言将C++中的数据结构、函数等包装成Python语言可调用的拓展接口模块，以此方法实现了运行速度与可扩展性的平衡。除此之外，CAVD中使用纯python代码实现了离子半径计算、阴离子堆积方式确定等模块。因此CAVD的源代码，主要分为C++源代码、Cython源代码和纯python代码。

# C++源代码：

## 与通道计算相关的代码

/\* 自定义异常 \*/

struct ZeoVectorException : public exception{

const char \* what () const throw (){

return "Exception: Pore basis vector is zero vector.";

}

};

struct IllogicalResultException : public exception{

const char \* what () const throw (){

return "Exception: Illogical result when attempting to identify channels/pockets.";

}

};

struct AccessibilityException : public exception{

const char \* what () const throw (){

return "Exception: Accessibility of node was determined more than once.";

}

};

bool CHANNEL::findChannels\_new(VORONOI\_NETWORK \*vornet, double minRadius, vector<CHANNEL> \*channels)

{

vector<bool> infoStorage;

try{

findChannels(vornet,minRadius,&infoStorage,channels);

}

catch (ZeoVectorException& e1){

cout << e1.what() << endl;

return false;

}

catch (IllogicalResultException& e2){

cout << e2.what() << endl;

return false;

}

catch (AccessibilityException& e3){

cout << e3.what() << endl;

return false;

}

return true;

}

/\*\* Write the commands necessary to draw the CHANNEL in ZeoVis

\* to the provided output stream. \*/

bool writeToVMD\_new(vector<CHANNEL> channels, char \*filename){

fstream output;

try{

output.open(filename, fstream::out);

for(unsigned int i = 0; i < channels.size(); i++){

channels.at(i).writeToVMD(i, output);

}

cout << "Writing ZeoVis information to .zchan file sucessful!" << endl;

return true;

}

catch (WritingCHANNELException& e1){

cout << e1.what() << endl;

return false;

}

}

/\*\* Write the CHANNEL to network file

\*/

void CHANNEL::writeToNET(int n, fstream &output){

if(!output.is\_open()){

cerr << "Error: File stream needed to print channel information was not open." << "\n"

<< "Exiting ..." << "\n";

throw WritingCHANNELException();

}

else{

output << "channeId " << n << "\n";

output << "dimensionality " << dimensionality << "\n";

// output << "(" << v\_a.x << "," << v\_a.y << "," << v\_a.z << ")\n";

// output << "(" << v\_b.x << "," << v\_b.y << "," << v\_b.z << ")\n";

// output << "(" << v\_c.x << "," << v\_c.y << "," << v\_c.z << ")\n";

output << "Interstitial table:" << "\n";

// Draw the components located in each unit cell

for(unsigned int i = 0; i < unitCells.size(); i++){

vector<int> nodeIDs = ucNodes.at(i);

DELTA\_POS disp = unitCells.at(i);

// Iterate over all nodes in the unit cell

for(unsigned int j = 0; j < nodeIDs.size(); j++){

DIJKSTRA\_NODE curNode = nodes.at(nodeIDs.at(j));

//output << j << "\t";

output << curNode.id << "\t";

//output << curNode.x << "\t" << curNode.y << "\t" << curNode.z << "\t";

//直角坐标

double xCoord = curNode.x + v\_a.x\*disp.x + v\_b.x\*disp.y + v\_c.x\*disp.z;

double yCoord = curNode.y + v\_a.y\*disp.x + v\_b.y\*disp.y + v\_c.y\*disp.z;

double zCoord = curNode.z + v\_a.z\*disp.x + v\_b.z\*disp.y + v\_c.z\*disp.z;

output << xCoord << "\t" << yCoord << "\t" << zCoord << "\t" << disp.x << "\t" << disp.y << "\t" << disp.z << "\t";

output << curNode.max\_radius << endl;

}

}

output << "\n" << "Connection table:" << "\n";

for(unsigned int i = 0; i < unitCells.size(); i++){

vector<int> nodeIDs = ucNodes.at(i);

DELTA\_POS disp = unitCells.at(i);

// Iterate over all nodes in the unit cell

for(unsigned int j = 0; j < nodeIDs.size(); j++){

DIJKSTRA\_NODE curNode = nodes.at(nodeIDs.at(j));

// Iterate over all connections stemming from the current node

for(unsigned int k = 0; k < curNode.connections.size(); k++){

CONN curConn = curNode.connections.at(k);

DIJKSTRA\_NODE otherNode = nodes.at(curConn.to);

output << curNode.id << "\t" << otherNode.id << "\t" << curConn.max\_radius << endl;

}

}

}

}

}

bool writeToNET\_new(vector<CHANNEL> channels, char \*filename){

fstream output;

try{

output.open(filename, fstream::out);

for(unsigned int i = 0; i < channels.size(); i++){

channels.at(i).writeToNET(i, output);

}

cout << "Writing CHANNEL information to .net file sucessful!" << endl;

return true;

}

catch (WritingCHANNELException& e1){

cout << e1.what() << endl;

return false;

}

}

## 与Voronoi网络分析相关的代码

/\* 自定义异常 \*/

struct InvalidParticlesNumException : public exception{

const char \* what () const throw (){

return "Exception: Invalid number of particles provided for Voronoi decomposition.";

}

};

struct InvalidBoxDimException : public exception{

const char \* what () const throw (){

return "Exception: valid box dimensions calculated for Voronoi decomposition.";

}

};

struct HugeGridException : public exception{

const char \* what () const throw (){

return "Exception: voro++: Number of computational blocks exceeds the maximum.";

}

};

struct AttemptException : public exception{

const char \* what () const throw (){

return "Exception: Attempt numbers larger than excepted.";

}

};

struct VoronoiDecompException : public exception{

const char \* what () const throw (){

return "Exception: Unable to make Voronoi decomposition.";

}

};

struct CoordNumException : public exception{

const char \* what () const throw (){

return "Exception: Improper number of node coordinates in Voronoi decomposition.";

}

};

// Voronoi分解函数

bool performVoronoiDecomp(bool radial, ATOM\_NETWORK \*atmnet, VORONOI\_NETWORK \*vornet, vector<VOR\_CELL> \*cells, bool saveVorCells,

vector<BASIC\_VCELL> \*bvcells){

container\_periodic\_poly \*rad\_con = NULL;

container\_periodic \*no\_rad\_con = NULL;

try{

if (radial)

rad\_con = (container\_periodic\_poly \*)performVoronoiDecomp(radial, atmnet, vornet, \*cells, saveVorCells, \*bvcells);

else

no\_rad\_con = (container\_periodic \*)performVoronoiDecomp (radial, atmnet, vornet, \*cells, saveVorCells, \*bvcells);

delete rad\_con;

delete no\_rad\_con;

return true;

}

catch (InvalidParticlesNumException& e1){

cout << e1.what() << endl;

return false;

}

catch (InvalidBoxDimException& e2){

cout << e2.what() << endl;

return false;

}

catch (HugeGridException& e3){

cout << e3.what() << endl;

return false;

}

catch (AttemptException& e4){

cout << e4.what() << endl;

return false;

}

catch (VoronoiDecompException& e5){

cout << e5.what() << endl;

return false;

}

catch (CoordNumException& e6){

cout << e6.what() << endl;

return false;

}

}

/\*\*

\* Structure a new function to return whether a specific radius atom can through voronoi network

\*/

//int throughVorNet(VORONOI\_NETWORK \*vornet, char\* filename, double migrantRad){

//bool throughVorNet(VORONOI\_NETWORK \*vornet, char\* filename, double \*Ri, double \*Rf, double \*Rif, double migrantRad){

bool throughVorNet(VORONOI\_NETWORK \*vornet, char\* filename, double \*Ri, double \*Rf, double \*Rif){

vector<double> freeRadResults;

vector<double> incRadResults;

vector<bool> NtoN;

double Di,Df,Dif;

DELTA\_POS directions [3] = {DELTA\_POS(1,0,0), DELTA\_POS(0,1,0), DELTA\_POS(0,0,1)};

for(unsigned int i = 0; i < 3; i++){

VORONOI\_NETWORK newNet;

set<int> sourceNodes;

map<int,int> idAliases;

extendVorNet(vornet, &newNet, directions[i], &idAliases, &sourceNodes);

DIJKSTRA\_NETWORK dnet;

DIJKSTRA\_NETWORK::buildDijkstraNetwork(&newNet,&dnet);

TRAVERSAL\_NETWORK analyzeNet = TRAVERSAL\_NETWORK(directions[i].x,directions[i].y,directions[i].z, &dnet);

pair<bool,PATH> results = analyzeNet.findMaxFreeSphere(&idAliases, &sourceNodes);

freeRadResults.push\_back(results.second.max\_radius);

incRadResults.push\_back(results.second.max\_inc\_radius);

NtoN.push\_back(results.first);

}

Di = findMaxIncludedSphere(vornet);

// output << filename << " " << "Di = " findMaxIncludedSphere(vornet) << " " << "Df = " ;

double maxd=0.0; int maxdir=0;

for(unsigned int i = 0; i < freeRadResults.size(); i++)

{

if(i==0) {maxd=freeRadResults[i]; maxdir=i;}

else

{

if(maxd<freeRadResults[i])

{

maxd=freeRadResults[i];

maxdir=i;

}

else if(maxd==freeRadResults[i])

{

if(incRadResults[maxdir]<incRadResults[i]) maxdir=i;

};

};

};

Df = freeRadResults[maxdir];

Dif = incRadResults[maxdir];

\*Ri = Di;

\*Rf = Df;

\*Rif = Dif;

fstream output;

output.setf(ios::fixed,ios::floatfield);

output.precision(5);

output.width(12);

output.open(filename, fstream::out);

output << filename << " " << Di << " " << Df << " " << Dif << "\n";

output.close();

cout << filename << " " << "Ri = " << Di << " " << "Rf = " << Df << " " << "Rif = " << Dif << endl;

}

## 与数据输入输出相关的代码

/\*\*

\* remove a migrant ion in .cif file and restore it as a new .cif files

\* added at 20180408

\* \*/

void string\_replace(string &strBig, const string &strsrc, const string &strdst){

string::size\_type pos = 0;

string::size\_type srclen = strsrc.size();

string::size\_type dstlen = strdst.size();

while((pos=strBig.find(strsrc, pos)) != string::npos){

strBig.replace( pos, srclen, strdst );

pos += dstlen;

}

}

string pretreatCifFilename(char \*filename,const char \*migrant){

string line;

int ndx;

vector<string> token;

// Try opening the file if it opens proceed with processing

ifstream ciffile;

cout << "Treat file: " << filename << endl;

ciffile.open(filename);

if(ciffile.is\_open()){

//output fstream

fstream output;

string str = (string)filename;

string outfilename = str.replace(str.find\_last\_of("."),4,"\_remove"+(string)migrant+".cif");

//outfilename = str.replace(outfilename.find\_last\_of("/"),1,"/results/");

cout << "outfilename = " << outfilename << endl;

output.open(outfilename.data(), fstream::out);

if(output.is\_open()){

//cout << "ifstream and fstream open!" << endl;

while(!ciffile.eof()){

getline(ciffile,line);

token = split(line," ()\r\t");

exception: //an easy way to jump out of the \_loop command if an unknown command was found

if (token.size() > 0){

//Where all non-loop commands should be added

if (token[0].compare("loop\_") == 0){ //loop\_

output << line << endl;

getline(ciffile,line);

token = split(line," \r\t");

bool tokenized = false, in\_loop = true;

if(token.size()>0) tokenized=true;

while (tokenized && in\_loop){ //check \_loop column label

if(token[0].at(0)=='\_') {

output << line << endl;

getline(ciffile,line);

token = split(line," \r\t");

if(token.size()<=0) tokenized=false;

}else in\_loop = false;

}

//cout<< "break cycle" << endl;

token = split(line," ,'\r\t"); //This is needed to split data

while(token.size()>0){

if (token[0].at(0) =='\_' || token[0].compare("loop\_")==0 || token[0].at(0) == '#'){

goto exception; // unexpected input

}

string::size\_type idx;

idx=line.find(migrant);

//check and find migrant ion

if(idx != string::npos){

//cout <<"detect migrant ion " << migrant << " and remove it." << endl;

getline(ciffile,line);

token = split(line," ,'\r\t");

}else {

output << line << endl;

getline(ciffile,line);

token = split(line," ,'\r\t");

}

}

if(token.size()==0) output << line << endl;

}else output << line << endl;

}else output << line << endl;

}

ciffile.close();

output.close();

cout << "Remove migrant ion completed!" << endl;

return outfilename;

}

else{

cerr << "Error: Failed to open .cif output file " << outfilename << endl;

return "";

}

}

else{

cerr << "Error: Failed to open .cif input file " << filename << endl;

return "";

}

}

/\*\* Write the information stored within the VORONOI\_NETWORK in a .BI

file format to the provided filename. Excludes any nodes or nodes with radii

less than the provided threshold. For the default 0（当minRad为默认值0时）, all nodes and edges are included\*/

bool writeToBI(char \*filename, ATOM\_NETWORK \*cell, VORONOI\_NETWORK \*vornet, double minRad){

fstream output;

double a,b,c;

output.open(filename, fstream::out);

if(!output.is\_open()){

cerr << "Error: Failed to open .net2 output file " << filename << "\n";

//cerr << "Exiting ..." << "\n";

//exit(1);

return false;

}

else{

cout << "Writing Bottleneck and Interstitial network information to " << filename << "\n";

// Write Voronoi node information

output << "Interstitial table:" << "\n";

vector<VOR\_NODE> ::iterator niter = vornet->nodes.begin();

int i = 0;

while(niter != vornet->nodes.end()){

if(niter->rad\_stat\_sphere > minRad){

a = niter->x \* cell->invUCVectors[0][0] + niter->y \* cell->invUCVectors[0][1] + niter->z \* cell->invUCVectors[0][2];

b = niter->y \* cell->invUCVectors[1][1] + niter->z \* cell->invUCVectors[1][2];

c = niter->z \* cell->invUCVectors[2][2];

output << i << " " << a << " " << b << " "

<< c << " " << niter->rad\_stat\_sphere;

output << "\n";

}

i++;

niter++;

}

// Write Voronoi edge information

output << "\n" << "Channel table:" << "\n";

vector<VOR\_EDGE> ::iterator eiter = vornet->edges.begin();

while(eiter != vornet->edges.end()){

if(eiter->rad\_moving\_sphere > minRad){

a = eiter->bottleneck\_x \* cell->invUCVectors[0][0] + eiter->bottleneck\_y \* cell->invUCVectors[0][1] + eiter->bottleneck\_z \* cell->invUCVectors[0][2];

b = eiter->bottleneck\_y \* cell->invUCVectors[1][1] + eiter->bottleneck\_z \* cell->invUCVectors[1][2];

c = eiter->bottleneck\_z \* cell->invUCVectors[2][2];

output << eiter->from << " -> " << eiter->to <<" "

<< a <<" " << b <<" " << c << " " << eiter->rad\_moving\_sphere << "\n";

}

eiter++;

}

}

output.close();

return true;

}

//Added at 20180420

/\*\*

\* write the bottleneck, interstitial and atomnetwork information to .vasp

\* file format to the provided filename. Excludes any nodes or nodes with radii

\* less than the provided threshold. For the default 0 minRad value, all nodes

\* and edges are included.\*/

bool writeToVasp(char \*filename, ATOM\_NETWORK \*cell, VORONOI\_NETWORK \*vornet, bool storeRadius, double minRad, double maxRad){

fstream output;

//string compareatom = cell->atoms.at(0).type;//用于比较原子类型

int atomcount = 0;//计数标志符

int flag = 0;

vector<string> atomtype;

vector<int> atomnum;

double a,b,c;//分数坐标

output.open(filename, fstream::out);

if(!output.is\_open()){

cerr << "Error: Failed to open .vasp output file " << filename << "\n";

//cerr << "Exiting ..." << "\n";

//exit(1);

return false;

}

else{

cout << "Writing structure information to " << filename << "\n";

// Write unit cell information

output << cell->name << "\n";

output << "1.0" << "\n";//缩放系数

output << " " << cell->v\_a.x << " " << cell->v\_a.y << " " << cell->v\_a.z << "\n";

output << " " << cell->v\_b.x << " " << cell->v\_b.y << " " << cell->v\_b.z << "\n";

output << " " << cell->v\_c.x << " " << cell->v\_c.y << " " << cell->v\_c.z << "\n";

//calculate the number of different atoms

atomtype.push\_back(cell->atoms.at(0).type);

for(int i = 0; i<cell->numAtoms; i++){

if(cell->atoms.at(i).type.compare(atomtype.at(flag)) == 0){

atomcount ++;

}

else{

atomnum.push\_back(atomcount);

flag++;

atomtype.push\_back(cell->atoms.at(i).type);

atomcount = 1;

}

if(i == cell->numAtoms -1){

atomnum.push\_back(atomcount);

atomcount = 0;//计数标志符归零

}

}

//calculate the interstitial numbers numbers

vector<VOR\_NODE> ::iterator niter = vornet->nodes.begin();

while(niter != vornet->nodes.end()){

if((minRad == 0.0 && maxRad == 0.0) || (niter->rad\_stat\_sphere >= minRad && niter->rad\_stat\_sphere <= maxRad))

atomcount++;

niter++;

}

atomtype.push\_back("It");//间隙符号

atomnum.push\_back(atomcount);

atomcount = 0;//计数标志符归零

//calculate the bottleneck numbers

vector<VOR\_EDGE> ::iterator eiter = vornet->edges.begin();

while(eiter != vornet->edges.end()){

if((minRad == 0.0 && maxRad == 0.0) ||(eiter->rad\_moving\_sphere >= minRad && eiter->rad\_moving\_sphere <= maxRad))

atomcount++;

eiter++;

}

atomtype.push\_back("Bn");//间隙符号

atomnum.push\_back(atomcount);

//Write static information

for(int i = 0; i< atomtype.size(); i++){

output << " " << atomtype[i];

}

output << "\n";

for(int i = 0; i< atomnum.size(); i++){

output << " " << atomnum[i];

}

output << "\n";

//write the unit cell coordinate information

output << "Direct" << "\n";

for(int i = 0; i<cell->numAtoms; i++){

output << " " << cell->atoms.at(i).a\_coord << " " << cell->atoms.at(i).b\_coord << " " << cell->atoms.at(i).c\_coord << "\n";

}

//wirte the interstitial informations

niter = vornet->nodes.begin();

while(niter != vornet->nodes.end()){

if((minRad == 0.0 && maxRad == 0.0) || (niter->rad\_stat\_sphere >= minRad && niter->rad\_stat\_sphere <= maxRad)){

//cell->initMatrices();

a = niter->x \* cell->invUCVectors[0][0] + niter->y \* cell->invUCVectors[0][1] + niter->z \* cell->invUCVectors[0][2];

b = niter->y \* cell->invUCVectors[1][1] + niter->z \* cell->invUCVectors[1][2];

c = niter->z \* cell->invUCVectors[2][2];

output <<" " << a << " " << b << " " << c <<"\n";

}

niter++;

}

// Write botttleneck information

eiter = vornet->edges.begin();

while(eiter != vornet->edges.end()){

if((minRad == 0.0 && maxRad == 0.0) || (eiter->rad\_moving\_sphere >= minRad && eiter->rad\_moving\_sphere <= maxRad)){

a = eiter->bottleneck\_x \* cell->invUCVectors[0][0] + eiter->bottleneck\_y \* cell->invUCVectors[0][1] + eiter->bottleneck\_z \* cell->invUCVectors[0][2];

b = eiter->bottleneck\_y \* cell->invUCVectors[1][1] + eiter->bottleneck\_z \* cell->invUCVectors[1][2];

c = eiter->bottleneck\_z \* cell->invUCVectors[2][2];

output <<" " << a << " " << b << " " << c <<"\n";

}

eiter++;

}

//write Radius information

if(storeRadius == true){

output << "Radius" << "\n";

for(int i = 0; i<cell->numAtoms; i++){

output << " " << cell->atoms.at(i).radius << "\n";

}

//wirte the interstitial radius informations

niter = vornet->nodes.begin();

while(niter != vornet->nodes.end()){

if((minRad == 0.0 && maxRad == 0.0) || (niter->rad\_stat\_sphere >= minRad && niter->rad\_stat\_sphere <= maxRad)){

output <<" " << niter->rad\_stat\_sphere <<"\n";

}

niter++;

}

// Write botttleneck radius information

eiter = vornet->edges.begin();

while(eiter != vornet->edges.end()){

if((minRad == 0.0 && maxRad == 0.0) || (eiter->rad\_moving\_sphere >= minRad && eiter->rad\_moving\_sphere <= maxRad)){

output <<" " << eiter->rad\_moving\_sphere <<"\n";

}

eiter++;

}

}

}

output.close();

return true;

}

//Added at 20180826

/\*\*

\* Write ATOM\_NETWORK to .vasp file

\*\*/

bool writeAtmntToVasp(char \*filename, ATOM\_NETWORK \*cell, bool storeRadius){

fstream output;

int atomcount = 0;//计数标志符

int flag = 0;

vector<string> atomtype;

vector<int> atomnum;

double a,b,c;//分数坐标

output.open(filename, fstream::out);

if(!output.is\_open()){

cerr << "Error: Failed to open .vasp output file " << filename << "\n";

//cerr << "Exiting ..." << "\n";

//exit(1);

return false;

}

else{

cout << "Writing ATOM\_NETWORK information to " << filename << "\n";

// Write unit cell information

output << cell->name << "\n";

output << "1.0" << "\n";//缩放系数

output << " " << cell->v\_a.x << " " << cell->v\_a.y << " " << cell->v\_a.z << "\n";

output << " " << cell->v\_b.x << " " << cell->v\_b.y << " " << cell->v\_b.z << "\n";

output << " " << cell->v\_c.x << " " << cell->v\_c.y << " " << cell->v\_c.z << "\n";

//calculate the number of different atoms

atomtype.push\_back(cell->atoms.at(0).type);

for(int i = 0; i<cell->numAtoms; i++){

if(cell->atoms.at(i).type.compare(atomtype.at(flag)) == 0){

atomcount ++;

}

else{

atomnum.push\_back(atomcount);

flag++;

atomtype.push\_back(cell->atoms.at(i).type);

atomcount = 1;

}

if(i == cell->numAtoms -1){

atomnum.push\_back(atomcount);

atomcount = 0;//计数标志符归零

}

}

//Write static information

for(int i = 0; i< atomtype.size(); i++){

output << " " << atomtype[i];

}

output << "\n";

for(int i = 0; i< atomnum.size(); i++){

output << " " << atomnum[i];

}

output << "\n";

//write the unit cell coordinate information

output << "Direct" << "\n";

for(int i = 0; i<cell->numAtoms; i++){

output << " " << cell->atoms.at(i).a\_coord << " " << cell->atoms.at(i).b\_coord << " " << cell->atoms.at(i).c\_coord << "\n";

}

//write Radius information

if(storeRadius == true){

output << "Radius" << "\n";

for(int i = 0; i<cell->numAtoms; i++){

output << " " << cell->atoms.at(i).radius << "\n";

}

}

}

output.close();

return true;

}

## 与ASA计算相关代码

void computeASA\_new(char \*filename, ATOM\_NETWORK \*atmnet, bool highAccuracy, double r\_probe\_chan, double r\_probe, int numSamples){

fstream output;

ATOM\_NETWORK orgAtomnet;

output.open(filename, fstream::out);

atmnet->copy(&orgAtomnet); // keep a copy of original atomnet (if high accuracy is not set, this is the same as analyzed network)

if(highAccuracy){

// atmnet.copy(&orgAtomnet); // keep a copy of original atomnet

// calling the following function will modify atmnet - replace large atoms with clusters of small ones

setupHighAccuracyAtomNetwork(atmnet, "LOW");

};

//calcASA(&atmnet, &orgAtomnet, highAccuracy, chan\_radius, probe\_radius, calcDensity(&atmnet), numSamples, true, output, (char \*)filename.data(), visualize, visVisITflag, LiverpoolFlag, ExtendedOutputFlag);

double area = calcASA(atmnet, &orgAtomnet, highAccuracy, r\_probe\_chan, r\_probe, calcDensity(atmnet), numSamples, true , output, filename, true, false, false, false);

output.close();

cout << "write information to .zsa file success!" << endl;

}

# Cython源代码（.pxd，.pyx）：

## 通道包装代码

**Channel.pxd**

from libcpp.vector cimport vector

from libcpp.map cimport map

from netstorage cimport VORONOI\_NETWORK

from graphstorage cimport DELTA\_POS

from graphstorage cimport CONN

from graphstorage cimport DIJKSTRA\_NODE

from graphstorage cimport DIJKSTRA\_NETWORK

from geometry cimport XYZ

cdef extern from "../../zeo++/channel.h":

cdef cppclass CHANNEL:

map[int,int] idMappings

map[int,int] reverseIDMappings

vector[DIJKSTRA\_NODE] nodes

vector[CONN] connections

vector[DELTA\_POS] unitCells

vector[vector[int]] ucNodes

XYZ v\_a, v\_b, v\_c

int dimensionality

CHANNEL() except +

cdef extern from "../../zeo++/channel.h" namespace "CHANNEL":

#cdef void findChannels(DIJKSTRA\_NETWORK\*, vector[bint] \*, vector[CHANNEL] \*)

cdef bint findChannels\_new(VORONOI\_NETWORK\*, double, vector[CHANNEL] \*)

cdef extern from "../../zeo++/channel.h":

cdef bint c\_writeToVMD "writeToVMD\_new"(vector[CHANNEL] channels, char \*filename)

cdef bint c\_writeToNET "writeToNET\_new"(vector[CHANNEL] channels, char \*filename)

cdef class Channel:

cdef CHANNEL\* thisptr

**channel.pyx**

# distutils: language = c++

# distutils: sources = ../channel.cc

#Added at 20180704

from cavd.netstorage cimport VoronoiNetwork

from cavd.netstorage cimport VORONOI\_NETWORK

from channel cimport CHANNEL

from graphstorage cimport DELTA\_POS

from graphstorage cimport CONN

from graphstorage cimport DIJKSTRA\_NODE

from graphstorage cimport DIJKSTRA\_NETWORK

from geometry cimport XYZ

from libcpp.vector cimport vector

#Added at 20180808

#Customize an exception class

class FindChannelError(Exception):

#print("Find Channel in Voronoi Network Failed!")

pass

cdef class Channel:

"""

Python wrapper to Zeo++ Channel.

"""

def \_\_cinit\_\_(self):

self.thisptr = new CHANNEL()

def \_\_dealloc\_\_(self):

del self.thisptr

property nodes:

def \_\_get\_\_(self):

nodes = []

cdef vector[DIJKSTRA\_NODE] dj\_nodes = self.thisptr.nodes

for i in range(dj\_nodes.size()):

dj\_id = dj\_nodes[i].id

dj\_coords = [dj\_nodes[i].x,dj\_nodes[i].y,dj\_nodes[i].z]

dj\_max\_radius = dj\_nodes[i].max\_radius

nodes.append([dj\_id, dj\_coords, dj\_max\_radius])

return nodes

property connections:

def \_\_get\_\_(self):

connections = []

cdef vector[CONN] conns = self.thisptr.connections

for i in range(conns.size()):

conn\_from = conns[i].origin

conn\_to = conns[i].ending

conn\_length = conns[i].length

conn\_max\_radius = conns[i].max\_radius

conn\_delta\_pos = [conns[i].deltaPos.x,conns[i].deltaPos.y,conns[i].deltaPos.z]

conn = [conn\_from, conn\_to, conn\_length, conn\_max\_radius, conn\_delta\_pos]

connections.append(conn)

return connections

property nodes\_deltapos:

def \_\_get\_\_(self):

nodes\_deltapos = []

cdef vector[DELTA\_POS] unitCells = self.thisptr.unitCells

cdef vector[vector[int]] ucNodes = self.thisptr.ucNodes

cdef DELTA\_POS pos

cdef vector[int] ucNode

for i in range(unitCells.size()):

pos = unitCells[i]

ucNode = ucNodes[i]

for j in range(ucNode.size()):

node\_id = ucNode[j]

node\_pos = [pos.x, pos.y, pos.z]

nodes\_deltapos.append([node\_id, node\_pos])

return nodes\_deltapos

property dimensionality:

def \_\_get\_\_(self):

return self.thisptr.dimensionality

property lattice:

def \_\_get\_\_(self):

la = [self.thisptr.v\_a.x, self.thisptr.v\_a.y, self.thisptr.v\_a.z]

lb = [self.thisptr.v\_b.x, self.thisptr.v\_b.y, self.thisptr.v\_b.z]

lc = [self.thisptr.v\_c.x, self.thisptr.v\_c.y, self.thisptr.v\_c.z]

lattice = [la, lb, lc]

return lattice

@classmethod

def findChannelsInVornet(cls, vornet, probe\_rad, filename):

cdef VORONOI\_NETWORK\* c\_vornet\_ptr = (<VoronoiNetwork?>vornet).thisptr

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

cdef char\* c\_filename = filename

cdef vector[CHANNEL] channels

if findChannels\_new(c\_vornet\_ptr, probe\_rad, &channels):

if not c\_writeToVMD(channels, c\_filename):

raise IOError

else:

raise FindChannelError

@classmethod

#Add at 20180826

def findChannels(cls, vornet, probe\_rad, filename):

cdef VORONOI\_NETWORK\* c\_vornet\_ptr = (<VoronoiNetwork?>vornet).thisptr

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

cdef char\* c\_filename = filename

cdef vector[CHANNEL] c\_channels

if not findChannels\_new(c\_vornet\_ptr, probe\_rad, &c\_channels):

raise FindChannelError

if not c\_writeToNET(c\_channels, c\_filename):

raise IOError

channels = []

channel = Channel()

for i in range(c\_channels.size()):

channel.thisptr.idMappings = (&(c\_channels[i])).idMappings

channel.thisptr.reverseIDMappings = (&(c\_channels[i])).reverseIDMappings

channel.thisptr.nodes = (&(c\_channels[i])).nodes

channel.thisptr.connections = (&(c\_channels[i])).connections

channel.thisptr.unitCells = (&(c\_channels[i])).unitCells

channel.thisptr.ucNodes = (&(c\_channels[i])).ucNodes

channel.thisptr.v\_a = (&(c\_channels[i])).v\_a

channel.thisptr.v\_b = (&(c\_channels[i])).v\_b

channel.thisptr.v\_c = (&(c\_channels[i])).v\_c

channel.thisptr.dimensionality = (&(c\_channels[i])).dimensionality

channels.append(channel)

return channels

## 图数据结构包装代码

**graphstorage.pxd**

# distutils: language = c++

# distutils: sources = ../graphstorage.cc

from libcpp.vector cimport vector

from netstorage cimport VORONOI\_NETWORK

from geometry cimport XYZ

cdef extern from "../../zeo++/graphstorage.h":

cdef cppclass DELTA\_POS:

int x, y, z

DELTA\_POS() except +

DELTA\_POS(int, int, int)

cdef cppclass CONN:

int origin "from"

int ending "to"

double length

double max\_radius

DELTA\_POS deltaPos

CONN() except +

CONN(int, int, double, double, int, int, int)

cdef cppclass DIJKSTRA\_NODE:

int id

double x, y, z

vector[CONN] connections

double max\_radius

bint active

DIJKSTRA\_NODE() except +

cdef cppclass DIJKSTRA\_NETWORK:

vector[DIJKSTRA\_NODE] nodes

XYZ v\_a, v\_b, v\_c

DIJKSTRA\_NETWORK() except +

cdef extern from "../../zeo++/graphstorage.h" namespace "DIJKSTRA\_NETWORK":

cdef void buildDijkstraNetwork(VORONOI\_NETWORK\*, DIJKSTRA\_NETWORK\*)

#cdef class DijkstraNode:

# """

# Cython wrapper class for Zeo++ DIJKSTRA\_NODE class.

# """

# cdef DIJKSTRA\_NODE\* thisptr

cdef class DeltaPos:

cdef DELTA\_POS\* thisptr

cdef class Conn:

cdef CONN\* thisptr

cdef class DijkstraNode:

cdef DIJKSTRA\_NODE\* thisptr

cdef class DijkstraNetwork:

"""

Cython wrapper class for Zeo++ DIJKSTRA\_NETWORK class.

"""

cdef DIJKSTRA\_NETWORK\* thisptr

**graphstorage.pyx**

# distutils: language = c++

# distutils: sources = ../graphstorage.cc

from netstorage cimport VoronoiNetwork

cdef class DijkstraNetwork:

"""

Python wrapper class to Zeo++ Djikstra Network

"""

#cdef DIJKSTRA\_NETWORK\* thisptr

def \_\_cinit\_\_(self):

self.thisptr = new DIJKSTRA\_NETWORK()

@classmethod

def from\_VoronoiNetwork(cls, vornet):

"""

Build Dijkstra Net from input Voronoi Net

"""

dijkstranet = DijkstraNetwork()

c\_vornet = (<VoronoiNetwork?>vornet).thisptr

buildDijkstraNetwork(c\_vornet, dijkstranet.thisptr)

return dijkstranet

def \_\_dealloc\_\_(self):

del self.thisptr

property lattice:

def \_\_get\_\_(self):

la = [self.thisptr.v\_a.x, self.thisptr.v\_a.y, self.thisptr.v\_a.z]

lb = [self.thisptr.v\_b.x, self.thisptr.v\_b.y, self.thisptr.v\_b.z]

lc = [self.thisptr.v\_c.x, self.thisptr.v\_c.y, self.thisptr.v\_c.z]

lattice = [la, lb, lc]

return lattice

property nodes:

def \_\_get\_\_(self):

nodes = []

cdef vector[DIJKSTRA\_NODE] c\_nodes = self.thisptr.nodes

for i in range(c\_nodes.size()):

node\_id = c\_nodes[i].id

node\_pos = [c\_nodes[i].x, c\_nodes[i].y, c\_nodes[i].z]

node\_radius = c\_nodes[i].max\_radius

c\_node\_conns = c\_nodes[i].connections

node\_conns = []

for i in range(c\_node\_conns.size()):

conn\_from = c\_node\_conns[i].origin

conn\_to = c\_node\_conns[i].ending

conn\_length = c\_node\_conns[i].length

conn\_max\_radius = c\_node\_conns[i].max\_radius

conn\_delta\_pos = [c\_node\_conns[i].deltaPos.x,c\_node\_conns[i].deltaPos.y,c\_node\_conns[i].deltaPos.z]

conn = [conn\_from, conn\_to, conn\_length, conn\_max\_radius, conn\_delta\_pos]

node\_conns.append(conn)

node = [node\_id, node\_pos, node\_radius, node\_conns]

nodes.append(node)

return nodes

cdef class DeltaPos:

def \_\_cinit\_\_(self, int x, int y, int z):

self.thisptr = new DELTA\_POS(x,y,z)

def \_\_init\_\_(self, int x, int y, int z):

pass

def \_\_dealloc\_\_(self):

del self.thisptr

property pos:

def \_\_get\_\_(self):

pos = [self.thisptr.x, self.thisptr.y, self.thisptr.z]

return pos

def \_\_set\_\_(self, pos): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.x = pos[0]

self.thisptr.y = pos[1]

self.thisptr.z = pos[2]

cdef class Conn:

def \_\_cinit\_\_(self, int origin, int ending, double length, double max\_radius, int x, int y, int z):

self.thisptr = new CONN(origin, ending, length, max\_radius, x, y, z)

def \_\_init\_\_(self, int origin, int ending, double length, double max\_radius, int x, int y, int z):

pass

def \_\_dealloc\_\_(self):

del self.thisptr

property origin:

def \_\_get\_\_(self):

origin = self.thisptr.origin

return origin

def \_\_set\_\_(self, origin): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.origin = origin

property ending:

def \_\_get\_\_(self):

ending = self.thisptr.ending

return ending

def \_\_set\_\_(self, ending): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.ending = ending

property length:

def \_\_get\_\_(self):

length = self.thisptr.length

return length

def \_\_set\_\_(self, length): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.length = length

property max\_radius:

def \_\_get\_\_(self):

max\_radius = self.thisptr.max\_radius

return max\_radius

def \_\_set\_\_(self, max\_radius): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.max\_radius = max\_radius

property pos:

def \_\_get\_\_(self):

pos = [self.thisptr.deltaPos.x, self.thisptr.deltaPos.y, self.thisptr.deltaPos.z]

return pos

def \_\_set\_\_(self, pos): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.deltaPos.x = pos[0]

self.thisptr.deltaPos.y = pos[1]

self.thisptr.deltaPos.z = pos[2]

cdef class DijkstraNode:

def \_\_cinit\_\_(self):

self.thisptr = new DIJKSTRA\_NODE()

def \_\_init\_\_(self):

pass

def \_\_dealloc\_\_(self):

del self.thisptr

property id:

def \_\_get\_\_(self):

id = self.thisptr.id

return id

def \_\_set\_\_(self, id): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.id = id

property coords:

def \_\_get\_\_(self):

coords = [self.thisptr.x, self.thisptr.y, self.thisptr.z]

return coords

def \_\_set\_\_(self, coords): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.x = coords[0]

self.thisptr.y = coords[1]

self.thisptr.z = coords[2]

property max\_radius:

def \_\_get\_\_(self):

max\_radius = self.thisptr.max\_radius

return max\_radius

def \_\_set\_\_(self, max\_radius): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.max\_radius = max\_radius

property connections:

def \_\_get\_\_(self):

cdef vector[CONN] conns = self.thisptr.connections

connections = []

for i in range(conns.size()):

conn\_from = conns[i].origin

conn\_to = conns[i].ending

conn\_length = conns[i].length

conn\_max\_radius = conns[i].max\_radius

conn\_delta\_pos = [conns[i].deltaPos.x,conns[i].deltaPos.y,conns[i].deltaPos.z]

conn = [conn\_from, conn\_to, conn\_length, conn\_max\_radius, conn\_delta\_pos]

connections.append(conn)

return connections

## 离子半径包装代码

**netinfo.pyx**

def readIonRadTable(ionicRadDic):

"""

Read Ionic radius values from input Dictionary

"""

cdef map[string, double] ionRadMap

cdef string c\_key

cdef double c\_value

for key in ionicRadDic:

c\_key = (<unicode>key).encode('utf8')

c\_value = ionicRadDic[key]

ionRadMap.insert(pair[string,double](c\_key,c\_value))

zeo\_readIonRadTable(ionRadMap)

def lookupIonRadius(element):

""""

Args:s

element:

Element name in conventional shorthand

Ex: Al for aluminum

Si for silicon

Returns:

radius of the input element

"""

#added at 20180606

if isinstance(element, unicode):

element = (<unicode>element).encode('utf8')

cdef string c\_element = element

radius = zeo\_lookupIonRadius(c\_element, True)

return radius

#Added at 20180627

def initializeIonRadTable():

"""

Populate the Goldschmidt Ion radius table with Zeo++ default values

"""

zeo\_initializeIonRadTable()

def readIonRadTableFile(filename):

"""

Read Ionic radii values from input file and replace the default values

"""

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

cdef char\* c\_filename = filename

zeo\_readIonRadTableFile(c\_filename)

## 输入输出包装代码

**netio.pxd**

# distutils: language = c++

# distutils: sources = ../networkio.cc

from cavd.netstorage cimport ATOM\_NETWORK, VORONOI\_NETWORK

from libcpp.string cimport string

#Added at 20180704

from libcpp.vector cimport vector

from cavd.voronoicell cimport VOR\_CELL, BASIC\_VCELL

from cavd.channel cimport CHANNEL

cdef extern from '../../zeo++/networkio.h':

# remove migrant ion added at 20180408

cdef string pretreatCifFilename(char \*filename,const char \*migrant)

# writeToBI

cdef bint writeToBI(char \*filename, ATOM\_NETWORK \*cell, VORONOI\_NETWORK \*vornet, double minRad)

cdef bint writeToBI(char \*filename, ATOM\_NETWORK \*cell, VORONOI\_NETWORK \*vornet)

# writeToVasp

#cdef bint writeToVasp(char \*filename, ATOM\_NETWORK \*cell, VORONOI\_NETWORK \*vornet, bint storeRadius, double minRad)

#edited at 20180530

cdef bint writeToVasp(char \*filename, ATOM\_NETWORK \*cell, VORONOI\_NETWORK \*vornet, bint storeRadius, double minRad, double maxRad)

cdef bint writeToVasp(char \*filename, ATOM\_NETWORK \*cell, VORONOI\_NETWORK \*vornet, bint storeRadius)

cdef bint writeAtmntToVasp(char \*filename, ATOM\_NETWORK \*cell, bint storeRadius)

# Added at 20180704

cdef extern from '../../zeo++/voronoicell.h':

cdef bint writeZVis(char \*filename, vector[VOR\_CELL] \*cells, ATOM\_NETWORK \*atmnet, VORONOI\_NETWORK \*vornet)

# At present the return value of performVoronoiDecomp is void\*

# Compile it after void\* is changed to bool in the original source file

cdef extern from "../../zeo++/network.h":

cdef bint performVoronoiDecomp(bint, ATOM\_NETWORK\*, VORONOI\_NETWORK\*,

vector[VOR\_CELL]\*, bint, vector[BASIC\_VCELL]\*)

# Add at 20180826

cdef extern from "../../zeo++/channel.h":

cdef bint c\_writeToVMD "writeToVMD\_new"(vector[CHANNEL] channels, char \*filename)

cdef bint c\_writeToNET "writeToNET\_new"(vector[CHANNEL] channels, char \*filename)

**netio.pyx**

# distutils: language = c++

# distutils: sources = ../networkio.cc

from netstorage cimport AtomNetwork, VoronoiNetwork

from netstorage cimport ATOM\_NETWORK, VORONOI\_NETWORK

#Added at 20180704

from libcpp.vector cimport vector

from cavd.voronoicell cimport VOR\_CELL, BASIC\_VCELL

from cavd.channel cimport CHANNEL

from cavd.netstorage import PerformVDError

from cavd.channel import Channel

# Define the python definitions for the zeo++ functions

def readCiffile(filename, radialflag):

atmnet = AtomNetwork()

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

cdef char\* c\_filename = filename

if not readCIFFile(c\_filename, atmnet.thisptr, radialflag):

raise ValueError # Find the appropriate error and return it

return atmnet

# write to .bi file. Added at 20180408

def writeBIFile(filename, atmnet, vornet, minRad = None):

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

cdef char\* c\_filename = filename

cdef ATOM\_NETWORK\* c\_atmnet = (<AtomNetwork?>atmnet).thisptr

cdef VORONOI\_NETWORK\* c\_vornet\_ptr = (<VoronoiNetwork?>vornet).thisptr

if minRad:

if not writeToBI(c\_filename, c\_atmnet, c\_vornet\_ptr, minRad):

raise IOError

else:

if not writeToBI(c\_filename, c\_atmnet, c\_vornet\_ptr):

raise IOError

# remove migrant ion. Added at 20180408

def getRemoveMigrantFilename(filename,migrant):

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

if isinstance(migrant, unicode):

migrant = (<unicode>migrant).encode('utf8')

cdef char\* c\_filename = filename

cdef const char\* c\_migrant = migrant

pretreatedFilename = pretreatCifFilename(c\_filename,c\_migrant)

if pretreatedFilename == "":

#raise IOError("Can't Open ", filename, " or Can't Write to outputfile.")

raise IOError

else:

return pretreatedFilename

# write to .vasp file. Added at 20180426

def writeVaspFile(filename, atmnet, vornet, storeRadius = False, minRad = None, maxRad = None):

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

cdef char\* c\_filename = filename

cdef ATOM\_NETWORK\* c\_atmnet = (<AtomNetwork?>atmnet).thisptr

cdef VORONOI\_NETWORK\* c\_vornet\_ptr = (<VoronoiNetwork?>vornet).thisptr

if storeRadius:

if minRad and maxRad:

if not writeToVasp(c\_filename, c\_atmnet, c\_vornet\_ptr, True, minRad, maxRad):

raise IOError

else:

if not writeToVasp(c\_filename, c\_atmnet, c\_vornet\_ptr, True):

raise IOError

else:

if minRad and maxRad:

if not writeToVasp(c\_filename, c\_atmnet, c\_vornet\_ptr, False, minRad, maxRad):

raise IOError

else:

if not writeToVasp(c\_filename, c\_atmnet, c\_vornet\_ptr, False):

raise IOError

# write to atomnetwork to .vasp file. Added at 20180827

def writeAtomNetVaspFile(filename, atmnet, storeRadius = False):

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

cdef char\* c\_filename = filename

cdef ATOM\_NETWORK\* c\_atmnet = (<AtomNetwork?>atmnet).thisptr

if not writeAtmntToVasp(c\_filename, c\_atmnet, storeRadius):

raise IOError

# write to .zvis file. In this file, all AtomNetwork and VoronoiNetwork information contained. Added at 20180704

def writeZVisFile(filename, rad\_flag, atmnet, vornet):

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

cdef char\* c\_filename = filename

cdef ATOM\_NETWORK\* c\_atmnet = (<AtomNetwork?>atmnet).thisptr

cdef VORONOI\_NETWORK\* c\_vornet\_ptr = (<VoronoiNetwork?>vornet).thisptr

cdef vector[VOR\_CELL] vcells

cdef vector[BASIC\_VCELL] bvcells

if not performVoronoiDecomp(rad\_flag, c\_atmnet, c\_vornet\_ptr, &vcells, True, &bvcells):

raise PerformVDError

if not writeZVis(c\_filename, &vcells, c\_atmnet, c\_vornet\_ptr):

raise IOError

## 网络数据结构包装代码

**netstorage.pxd**

# distutils: language = c++

# distutils: sources = ../networkstorage.cc

from libcpp.vector cimport vector

from libcpp.string cimport string

from cavd.geometry cimport CPoint

from cavd.voronoicell cimport VOR\_CELL, BASIC\_VCELL

from cavd.geometry cimport XYZ

cdef extern from "../../zeo++/networkstorage.h":

cdef cppclass ATOM:

ATOM() except +

double x, y, z

double a\_coord, b\_coord, c\_coord

double radius

string atom\_type "type"

string label

int specialID

double mass

double charge

cdef cppclass ATOM\_NETWORK:

ATOM\_NETWORK() except +

void copy(ATOM\_NETWORK\*)

double a, b, c # Lattice parameters

double alpha, beta, gamma # lattice angles

XYZ v\_a, v\_b, v\_c

int no\_atoms "numAtoms"

vector[ATOM] atoms

CPoint abc\_to\_xyz(double, double, double)

CPoint abc\_to\_xyz(CPoint)

CPoint xyz\_to\_abc(double, double, double)

CPoint xyz\_to\_abc(CPoint)

cdef cppclass VOR\_NODE:

VOR\_NODE() except +

VOR\_NODE(double, double, double, double, vector[int])

double x, y, z

double rad\_stat\_sphere

cdef cppclass VOR\_EDGE:

VOR\_EDGE() except +

VOR\_EDGE(int, int, double, int, int, int, double)

#Added at 20180408

VOR\_EDGE(int, int, double, double, double, double, int, int, int, double)

int origin "from"

int ending "to"

double rad\_moving\_sphere

double length

int delta\_uc\_x, delta\_uc\_y, delta\_uc\_z

#added at 20180408

double bottleneck\_x,bottleneck\_y,bottleneck\_z;

cdef cppclass VORONOI\_NETWORK:

VORONOI\_NETWORK() except +

VORONOI\_NETWORK prune(double)

vector[VOR\_NODE] nodes

vector[VOR\_EDGE] edges

cdef bint c\_substituteAtoms "substituteAtoms"(ATOM\_NETWORK\*, ATOM\_NETWORK\*,

bint, int\*, bint)

cdef bint c\_fracSubstituteAtoms "fracSubstituteAtoms"(ATOM\_NETWORK\*,

ATOM\_NETWORK\*, bint, double,

int, int\*, double\*, bint)

cdef extern from '../../zeo++/networkio.h':

cdef bint writeAtmntToVasp(char \*filename, ATOM\_NETWORK \*cell, bint storeRadius)

# At present the return value of performVoronoiDecomp is void\*

# Compile it after void\* is changed to bool in the original source file

cdef extern from "../../zeo++/network.h":

cdef bint performVoronoiDecomp(bint, ATOM\_NETWORK\*, VORONOI\_NETWORK\*,

vector[VOR\_CELL]\*, bint, vector[BASIC\_VCELL]\*)

cdef void calculateFreeSphereParameters(VORONOI\_NETWORK\*, char\*, bint)

cdef void calculateConnParameters(VORONOI\_NETWORK \*, char \*, vector[double] \*)

cdef bint throughVorNet(VORONOI\_NETWORK\*, char\*, double\*, double\*, double\*)

cdef extern from "../../zeo++/area\_and\_volume.h":

cdef void visVoro(char\* name, double probeRad, int skel\_a, int skel\_b, int skel\_c,

VORONOI\_NETWORK\* vornet, ATOM\_NETWORK\* atmnet)

cdef class Atom:

"""

Cython wrapper class for Zeo++ ATOM class.

"""

cdef ATOM\* thisptr

cdef class AtomNetwork:

"""

Cython wrapper class for Zeo++ ATOM\_NETWORK class.

Contains a pointer to ATOM\_NETWORK and a flag denoting whether radius

for each atomic species is non-zero.

"""

cdef ATOM\_NETWORK\* thisptr

cdef bint rad\_flag

cdef class VoronoiNode:

"""

Cython wrapper class for Zeo++ VOR\_NODE class.

"""

cdef VOR\_NODE\* thisptr

cdef class VoronoiNetwork:

"""

Cython wrapper class for Zeo++ VORONOI\_NETWORK class.

"""

cdef VORONOI\_NETWORK\* thisptr

cdef class VoronoiEdge:

"""

Cython wrapper class for Zeo++ VOR\_EDGE class.

"""

cdef VOR\_EDGE\* thisptr

**netstorage.pyx**

"""

Cython file defining methods for AtomNetwork and VoronoiNetowrk

declared in netstorage.pxd file.

"""

from libcpp.vector cimport vector

from cython.operator cimport dereference as deref, preincrement as inc

cimport cavd.netinfo

from cavd.voronoicell cimport VOR\_CELL, BASIC\_VCELL, VOR\_FACE

from cavd.geometry cimport CPoint, Point

#Added at 20180606

from libcpp.map cimport map

from libcpp.pair cimport pair

from libcpp.string cimport string

#STUFF='Hi'

#Added at 20180807

#Customize an exception class

class PerformVDError(Exception):

#print("Perform Voronoi Decompition failured!")

pass

cdef class Atom:

"""

Class to store the information about atom (or ion) in a structure.

"""

def \_\_cinit\_\_(self):

self.thisptr = new ATOM()

def \_\_init\_\_(self):

pass

def \_\_dealloc\_\_(self):

del self.thisptr

property cart\_coords:

def \_\_get\_\_(self):

coords = [self.thisptr.x, self.thisptr.y, self.thisptr.z]

return coords

def \_\_set\_\_(self, coords): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.x = coords[0]

self.thisptr.y = coords[1]

self.thisptr.z = coords[2]

property frac\_coords:

def \_\_get\_\_(self):

coords = [self.thisptr.a\_coord, self.thisptr.b\_coord, self.thisptr.c\_coord]

return coords

def \_\_set\_\_(self, coords): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.a\_coord = coords[0]

self.thisptr.b\_coord = coords[1]

self.thisptr.c\_coord = coords[2]

property radius:

def \_\_get\_\_(self): return self.thisptr.radius

def \_\_set\_\_(self, radius):

print ("This value is not supposed to be modified")

self.thisptr.radius = radius

property atom\_type:

def \_\_get\_\_(self): return self.thisptr.atom\_type.decode('utf-8')

def \_\_set\_\_(self, atom\_type):

print ("This value is not supposed to be modified")

self.thisptr.atom\_type = atom\_type

property label:

def \_\_get\_\_(self): return self.thisptr.label.decode('utf-8')

def \_\_set\_\_(self, label):

print ("This value is not supposed to be modified")

self.thisptr.label = label

property specialID:

def \_\_get\_\_(self): return self.thisptr.specialID

def \_\_set\_\_(self, specialID):

print ("This value is not supposed to be modified")

self.thisptr.specialID = specialID

property mass:

def \_\_get\_\_(self): return self.thisptr.mass

def \_\_set\_\_(self, mass):

print ("This value is not supposed to be modified")

self.thisptr.mass = mass

property charge:

def \_\_get\_\_(self): return self.thisptr.charge

def \_\_set\_\_(self, charge):

print ("This value is not supposed to be modified")

self.thisptr.charge = charge

cdef class AtomNetwork:

"""

Class to store and manipulate the input atom network.

"""

#Cython wrapper for Zeo++ ATOM\_NETWORK class.

#Contains a pointer to ATOM\_NETWORK and a flag denoting whether radius

#for each atomic species is non-zero.

def \_\_cinit\_\_(self):

self.thisptr = new ATOM\_NETWORK()

def \_\_init\_\_(self):

pass

def \_\_dealloc\_\_(self):

del self.thisptr

property lattice\_para:

def \_\_get\_\_(self):

return [self.thisptr.a, self.thisptr.b, self.thisptr.c]

property lattice\_angle:

def \_\_get\_\_(self):

return [self.thisptr.alpha, self.thisptr.beta, self.thisptr.gamma]

property lattice:

def \_\_get\_\_(self):

la = [self.thisptr.v\_a.x, self.thisptr.v\_a.y, self.thisptr.v\_a.z]

lb = [self.thisptr.v\_b.x, self.thisptr.v\_b.y, self.thisptr.v\_b.z]

lc = [self.thisptr.v\_c.x, self.thisptr.v\_c.y, self.thisptr.v\_c.z]

lattice = [la, lb, lc]

return lattice

property atoms\_num:

def \_\_get\_\_(self): return self.thisptr.no\_atoms

def \_\_set\_\_(self, atoms\_num):

print ("This value is not supposed to be modified")

self.thisptr.no\_atoms = atoms\_num

property atoms:

def \_\_get\_\_(self):

atoms = []

cdef vector[ATOM] c\_atoms = self.thisptr.atoms

for i in range(c\_atoms.size()):

atom\_type = c\_atoms[i].atom\_type.decode('utf-8')

#atom\_coords = [c\_atoms[i].a\_coord,c\_atoms[i].b\_coord,c\_atoms[i].c\_coord]

atom\_coords = [c\_atoms[i].x,c\_atoms[i].y,c\_atoms[i].z]

atoms.append([atom\_type, atom\_coords])

return atoms

def copy(self):

"""

Create a copy of the AtomNetwork instance

"""

newatmnet = AtomNetwork()

self.thisptr.copy(newatmnet.thisptr)

newatmnet.rad\_flag = self.rad\_flag

return newatmnet

# write to atomnetwork to .vasp file. Added at 20180827

def writeAtomNetVaspFile(self, filename, storeRadius = False):

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

cdef char\* c\_filename = filename

if not writeAtmntToVasp(c\_filename, self.thisptr, storeRadius):

raise IOError

#Added at 20180420

def through\_VorNet(self, filename):

"""

Computes the diameters of the largest included sphere, free sphere

and included sphere along free sphere path.

Arguments:

filename:

Name of file where the diameters are stored.

"""

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

#if isinstance(migrantRad, unicode):

# migrantRad = (<unicode>migrantRad).encode('utf8')

vornet, edge\_centers, face\_centers = self.perform\_voronoi\_decomposition(False)

cdef char\* c\_fname = filename

#cdef double c\_migrantRad = migrantRad

#Added at 20180530

cdef double\* c\_Ri\_ptr

cdef double\* c\_Rf\_ptr

cdef double\* c\_Rif\_ptr

cdef double c\_Ri,c\_Rf,c\_Rif

vornet\_ptr = (<VoronoiNetwork?>vornet).thisptr

if throughVorNet(vornet\_ptr, c\_fname, &c\_Ri, &c\_Rf, &c\_Rif):

#return True

#edited at 20180530

return c\_Ri,c\_Rf,c\_Rif

cdef class VoronoiNode:

"""

Class to store the voronoi nodes with coordinates and radius

"""

def \_\_cinit\_\_(self):

self.thisptr = new VOR\_NODE()

def \_\_init\_\_(self):

pass

def \_\_dealloc\_\_(self):

del self.thisptr

property coords:

def \_\_get\_\_(self):

coords = [self.thisptr.x, self.thisptr.y, self.thisptr.z]

return coords

def \_\_set\_\_(self, coords): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.x = coords[0]

self.thisptr.y = coords[1]

self.thisptr.z = coords[2]

property radius:

def \_\_get\_\_(self): return self.thisptr.rad\_stat\_sphere

def \_\_set\_\_(self, rad):

print ("This value is not supposed to be modified")

self.thisptr.rad\_stat\_sphere = rad

cdef class VoronoiEdge:

"""

Class to store the voronoi edges with some atrribute

"""

def \_\_cinit\_\_(self):

self.thisptr = new VOR\_EDGE()

def \_\_init\_\_(self):

pass

def \_\_dealloc\_\_(self):

del self.thisptr

property origin:

def \_\_get\_\_(self): return self.thisptr.origin

def \_\_set\_\_(self, origin):

print ("This value is not supposed to be modified")

self.thisptr.origin = origin

property ending:

def \_\_get\_\_(self): return self.thisptr.ending

def \_\_set\_\_(self, ending):

print ("This value is not supposed to be modified")

self.thisptr.ending = ending

property radius:

def \_\_get\_\_(self): return self.thisptr.rad\_moving\_sphere

def \_\_set\_\_(self, rad):

print ("This value is not supposed to be modified")

self.thisptr.rad\_moving\_sphere = rad

property leng:

def \_\_get\_\_(self): return self.thisptr.length

def \_\_set\_\_(self, length):

print ("This value is not supposed to be modified")

self.thisptr.length = length

property delta\_uc:

def \_\_get\_\_(self):

delta\_uc = [self.thisptr.delta\_uc\_x, self.thisptr.delta\_uc\_y, self.thisptr.delta\_uc\_z]

return delta\_uc

def \_\_set\_\_(self, delta\_uc): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.delta\_uc\_x = delta\_uc[0]

self.thisptr.delta\_uc\_y = delta\_uc[1]

self.thisptr.delta\_uc\_z = delta\_uc[2]

property bot\_coords:

def \_\_get\_\_(self):

bot\_coords = [self.thisptr.bottleneck\_x, self.thisptr.bottleneck\_y, self.thisptr.bottleneck\_z]

return bot\_coords

def \_\_set\_\_(self, coords): # Don't set this

"""

This variable is not supposed to be modified manually

"""

print ("This value is not supposed to be modified")

self.thisptr.bottleneck\_x = coords[0]

self.thisptr.bottleneck\_y = coords[1]

self.thisptr.bottleneck\_z = coords[2]

cdef class VoronoiNetwork:

"""

Class to store the Voronoi network generated from Voronoi decomposition

of atom network.

"""

#Cython wrapper for Zeo++ VORONOI\_NETWORK class.

#Contains a pointer to ATOM\_NETWORK and a flag denoting whether radisu

#for each atomic species is non-zero.

def \_\_cinit\_\_(self):

self.thisptr = new VORONOI\_NETWORK()

def \_\_init\_\_(self):

pass

def \_\_dealloc\_\_(self):

del self.thisptr

def size(self):

return self.thisptr.nodes.size()

property nodes:

def \_\_get\_\_(self):

nodes = []

cdef vector[VOR\_NODE] c\_nodes = self.thisptr.nodes

for i in range(c\_nodes.size()):

node\_coords = [c\_nodes[i].x,c\_nodes[i].y,c\_nodes[i].z]

node\_radius = c\_nodes[i].rad\_stat\_sphere

nodes.append([node\_coords, node\_radius])

return nodes

property edges:

def \_\_get\_\_(self):

edges = []

cdef vector[VOR\_EDGE] c\_edges = self.thisptr.edges

for i in range(c\_edges.size()):

edge\_origin = c\_edges[i].origin

edge\_ending = c\_edges[i].ending

edge\_radius = c\_edges[i].rad\_moving\_sphere

edge\_length = c\_edges[i].length

edge\_boltpos = [c\_edges[i].bottleneck\_x,c\_edges[i].bottleneck\_y,c\_edges[i].bottleneck\_z]

edges.append([edge\_origin, edge\_ending, edge\_radius, edge\_length, edge\_boltpos])

return edges

def connection\_values(filename, vornet):

"""

Computes the Radius of the largest included sphere, free sphere

and included sphere along free sphere path.

Arguments:

filename:

Name of file where the radius are stored.

"""

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

vornet\_ptr = (<VoronoiNetwork?>vornet).thisptr

cdef char\* c\_fname = filename

cdef double c\_Ri,c\_Rf,c\_Rif

throughVorNet(vornet\_ptr, c\_fname, &c\_Ri, &c\_Rf, &c\_Rif)

return c\_Ri,c\_Rf,c\_Rif

def connection\_values\_list(filename, vornet):

conn\_values = []

if isinstance(filename, unicode):

filename = (<unicode>filename).encode('utf8')

vornet\_ptr = (<VoronoiNetwork?>vornet).thisptr

cdef char\* c\_fname = filename

cdef vector[double] values

calculateConnParameters(vornet\_ptr, c\_fname, &values)

conn\_values = []

for i in range(values.size()):

conn\_values.append(values[i])

return conn\_values

# Python源代码（.py文件）

## 接口源代码（\_\_init\_\_.py）

import os

import sys

import re

from cavd.netstorage import AtomNetwork

from cavd.netstorage import connection\_values\_list

from cavd.channel import Channel

from cavd.area\_volume import asa\_new

from cavd.netio import \*

from cavd.ionic\_radii import get\_ionic\_radii

from pymatgen.core.structure import Structure

from pymatgen.analysis.local\_env import ValenceIonicRadiusEvaluator

#获取特定结构中离子的有效半径

def EffectiveRadCom(filename):

# stru = Structure.from\_file(filename)

# val\_eval = ValenceIonicRadiusEvaluator(stru)

# radii = val\_eval.radii

radii = get\_ionic\_radii(filename)

radii\_keys = list(radii.keys())

#为了防止保存的半径信息无法匹配此处对半径信息做特殊处理，如Ag+的半径会保存为Ag、Ag+、Ag1+1

for key in radii\_keys:

radii[re.sub('[^a-zA-Z]','',key)] = radii[key]

if re.search('[A-Z][a-z]\*\+|[A-Z][a-z]\*\-', key) != None:

s1 = re.sub('\+','1+',key)

radii[re.sub('\-','1-',s1)] = radii[key]

print(radii)

return radii

def AllCom(filename, probe\_rad, num\_sample, migrant=None, rad\_flag=True, effective\_rad=True, rad\_file=None, rad\_store\_in\_vasp=True, minRad=0.0, maxRad=0.0):

radii = {}

if rad\_flag and effective\_rad:

radii = EffectiveRadCom(filename)

if migrant:

remove\_filename = getRemoveMigrantFilename(filename,migrant)

else:

remove\_filename = filename

atmnet = AtomNetwork.read\_from\_CIF(remove\_filename, radii, rad\_flag, rad\_file)

if migrant:

os.remove(remove\_filename)

vornet,edge\_centers,fcs = atmnet.perform\_voronoi\_decomposition(False)

prefixname = filename.replace(".cif","")

writeBIFile(prefixname+"\_orgin.bi",atmnet,vornet)

writeVaspFile(prefixname+"\_orgin.vasp",atmnet,vornet,rad\_store\_in\_vasp)

writeVaspFile(prefixname+"\_selected.vasp",atmnet,vornet,rad\_store\_in\_vasp,minRad,maxRad)

conn = connection\_values\_list(prefixname+".resex", vornet)

oneD,twoD,threeD = ConnStatus(probe\_rad, conn)

Channel.findChannels(vornet,probe\_rad,prefixname+".net")

asa\_new(prefixname+".zsa",False,atmnet,probe\_rad,probe\_rad,num\_sample)

writeZVisFile(prefixname+".zvis", rad\_flag, atmnet, vornet)

return conn,oneD,twoD,threeD

#计算指定结构的瓶颈和间隙

def BIComputation(filename, migrant=None, rad\_flag=True, effective\_rad=True, rad\_file=None, rad\_store\_in\_vasp=True, minRad=0.0, maxRad=0.0):

radii = {}

if rad\_flag and effective\_rad:

radii = EffectiveRadCom(filename)

if migrant:

remove\_filename = getRemoveMigrantFilename(filename,migrant)

else:

remove\_filename = filename

atmnet = AtomNetwork.read\_from\_CIF(remove\_filename, radii, rad\_flag, rad\_file)

vornet,edge\_centers,fcs = atmnet.perform\_voronoi\_decomposition(False)

#delete temp file

if migrant:

os.remove(remove\_filename)

prefixname = filename.replace(".cif","")

writeBIFile(prefixname+"\_orgin.bi",atmnet,vornet)

writeVaspFile(prefixname+"\_orgin.vasp",atmnet,vornet,rad\_store\_in\_vasp)

writeVaspFile(prefixname+"\_selected.vasp",atmnet,vornet,rad\_store\_in\_vasp,minRad,maxRad)

#计算指定结构最大自由球体半径，最大包含球体半径和沿着最大自由球体路径上的最大包含球体半径：Rf Ri Rif

def ConnValCom(filename, migrant=None, rad\_flag=True, effective\_rad=True, rad\_file=None):

radii = {}

if rad\_flag and effective\_rad:

radii = EffectiveRadCom(filename)

if migrant:

remove\_filename = getRemoveMigrantFilename(filename,migrant)

else:

remove\_filename = filename

atmnet = AtomNetwork.read\_from\_CIF(remove\_filename, radii, rad\_flag, rad\_file)

if migrant:

os.remove(remove\_filename)

prefixname = filename.replace(".cif","")

Ri,Rf,Rif = atmnet.through\_VorNet(prefixname+".res")

return Ri,Rf,Rif

#计算某个结构的连通数值列表，存放a，b，c方向上的Rf

def ConnValListCom(filename, migrant=None, rad\_flag=True, effective\_rad=True, rad\_file=None):

radii = {}

if rad\_flag and effective\_rad:

radii = EffectiveRadCom(filename)

if migrant:

remove\_filename = getRemoveMigrantFilename(filename,migrant)

else:

remove\_filename = filename

atmnet = AtomNetwork.read\_from\_CIF(remove\_filename, radii, rad\_flag, rad\_file)

vornet,edge\_centers,fcs = atmnet.perform\_voronoi\_decomposition(False)

if migrant:

os.remove(remove\_filename)

prefixname = filename.replace(".cif","")

conn = connection\_values\_list(prefixname+".resex",vornet)

return conn

#判断某个结构的连通性,给定目标离子的半径，判断它是否是1D，2D，3D导通

def ConnStatusCom(filename, radius, migrant=None, rad\_flag=True, effective\_rad=True, rad\_file=None):

connlist = ConnValListCom(filename, migrant, rad\_flag, effective\_rad, rad\_file)

oneD = False

twoD = False

threeD = False

af = connlist[0]

bf = connlist[1]

cf = connlist[2]

if radius <= af:

aconn = True

else:

aconn = False

if radius <= bf:

bconn = True

else:

bconn = False

if radius <= cf:

cconn = True

else:

cconn = False

if aconn and bconn and cconn:

threeD = True

if (aconn and bconn) or (aconn and cconn) or (bconn and cconn):

twoD = True

if aconn or bconn or cconn:

oneD = True

return oneD,twoD,threeD

#根据连通数值列表，判断某个结构的连通性。给定一个原子的半径，判断它是否是1D，2D，3D导通

def ConnStatus(radius,connlist):

oneD = False

twoD = False

threeD = False

af = connlist[0]

bf = connlist[1]

cf = connlist[2]

if radius <= af:

aconn = True

else:

aconn = False

if radius <= bf:

bconn = True

else:

bconn = False

if radius <= cf:

cconn = True

else:

cconn = False

if aconn and bconn and cconn:

threeD = True

if (aconn and bconn) or (aconn and cconn) or (bconn and cconn):

twoD = True

if aconn or bconn or cconn:

oneD = True

return oneD,twoD,threeD

#计算通道

def ChannelCom(filename, probe\_rad, migrant=None, rad\_flag=True, effective\_rad=True, rad\_file=None):

radii = {}

if rad\_flag and effective\_rad:

radii = EffectiveRadCom(filename)

if migrant:

remove\_filename = getRemoveMigrantFilename(filename,migrant)

else:

remove\_filename = filename

atmnet = AtomNetwork.read\_from\_CIF(remove\_filename, radii, rad\_flag, rad\_file)

vornet,edge\_centers,fcs = atmnet.perform\_voronoi\_decomposition(False)

if migrant:

os.remove(remove\_filename)

prefixname = filename.replace(".cif","")

#Channel.findChannelsInVornet(vornet,probe\_rad,prefixname+".zchan")

Channel.findChannels(vornet,probe\_rad,prefixname+".net")

#计算ASA

def ASACom(filename, probe\_rad, num\_sample, migrant=None, rad\_flag=True, effective\_rad=True, rad\_file=None):

radii = {}

if rad\_flag and effective\_rad:

radii = EffectiveRadCom(filename)

if migrant:

remove\_filename = getRemoveMigrantFilename(filename,migrant)

else:

remove\_filename = filename

atmnet = AtomNetwork.read\_from\_CIF(remove\_filename, radii, rad\_flag, rad\_file)

if migrant:

os.remove(remove\_filename)

prefixname = filename.replace(".cif","")

asa\_new(prefixname+".zsa",False,atmnet,probe\_rad,probe\_rad,num\_sample)

#计算空隙网络

def VoidNetCom(filename, migrant=None, rad\_flag=True, effective\_rad=True, rad\_file=None):

radii = {}

if rad\_flag and effective\_rad:

radii = EffectiveRadCom(filename)

if migrant:

remove\_filename = getRemoveMigrantFilename(filename,migrant)

else:

remove\_filename = filename

atmnet = AtomNetwork.read\_from\_CIF(remove\_filename, radii, rad\_flag, rad\_file)

vornet,edge\_centers,fcs = atmnet.perform\_voronoi\_decomposition(False)

if migrant:

os.remove(remove\_filename)

prefixname = filename.replace(".cif","")

writeZVisFile(prefixname+".zvis", False, atmnet, vornet)

## 离子半径计算代码（ionic\_radii.py）

'''

离子半径计算程序。

修改日期：20180710

作者：YAJ

配位数计算原理：

以目标离子为中心，10A为半径画球形区域，得到区域内的离子坐标。之后，按照距离目标离子中心的距离排序并存入一个列表中。

从前往后遍历列表，判断每个离子的正负，直到出现第一个同号离子为止，此时记录下的异号离子数目即为配位数。

离子半径计算原理：

计算给定结构文件中对应位点的配位数后，依据元素、价态和配位数三者的信息查香农1976年有效离子半径表，得到对应的半径。

输入：结构文件

离子半径表：香农1976年离子半径。(ionic\_radii.json)

输出：

格式一：自定义数据结构Radius组成的列表。Radius包含 label、species、配位数、半径等信息。

模式二：列表。列表中的每个元素按照（ label、species、配位数、半径等信息）排列。

环境要求：

需安装pymatgen包

需将ionic\_radii.json文件放置到同与该文件（ionic\_radii.py）同一文件目录下。

使用方法：

更改“if \_\_name\_\_ == "\_\_main\_\_":”模块中调用的“get\_ionic\_radii()”中的参数为需要计算的cif文件。

程序目前的缺陷：

无法计算具有部分占据的结构文件。

无法计算香农1976年有效离子半径表中不包含的离子半径。

仅适用离子晶体计算。

'''

import os

import json

import math

import six

import warnings

from pymatgen.core.structure import Structure

from pymatgen.analysis.local\_env import VoronoiNN

from pymatgen.analysis.local\_env import ValenceIonicRadiusEvaluator

from bisect import bisect\_left

from pymatgen.io.cif import CifFile

from pymatgen.io.cif import CifParser

from pymatgen.io.cif import str2float

from monty.io import zopen

from six.moves import zip, cStringIO

import numpy as np

from itertools import groupby

from pymatgen.util.coord import in\_coord\_list\_pbc, find\_in\_coord\_list\_pbc

from pymatgen.core.composition import Composition

from pymatgen.electronic\_structure.core import Magmom

from pymatgen.core.operations import MagSymmOp

from collections import OrderedDict

from pymatgen.core.periodic\_table import Element, Specie, get\_el\_sp, DummySpecie

file\_dir = os.path.dirname(\_\_file\_\_)

#file\_dir = os.path.abspath("/home/yeanjiang/yaj/CAVD/pyCavd/zeo/")

rad\_file = os.path.join(file\_dir, 'ionic\_radii.json')

with open(rad\_file, 'r') as fp:

\_ion\_radii = json.load(fp)

# pymtgen实现的求离子半径

def get\_ionic\_radii\_pymatgen(filename):

stru = Structure.from\_file(filename)

val\_eval = ValenceIonicRadiusEvaluator(stru)

radii = val\_eval.radii

print("pymatgen radii:",radii)

class VoronoiNN\_self(VoronoiNN):

def \_\_init\_\_(self, tol=0, targets=None, cutoff=10.0,

allow\_pathological=False, weight='solid\_angle',

extra\_nn\_info=True):

super(VoronoiNN\_self, self).\_\_init\_\_()

self.tol = tol

self.cutoff = cutoff

self.allow\_pathological = allow\_pathological

self.targets = targets

self.weight = weight

self.extra\_nn\_info = extra\_nn\_info

#根据目标离子设定范围球形区域内的异号离子数量确定配位数，考虑周期性情况

def get\_cn(self, structure, n):

#print(structure)

center = structure[n]

neighbors = structure.get\_sites\_in\_sphere(center.coords, self.cutoff)

neighbors = [i[0] for i in sorted(neighbors, key=lambda s: s[1])]

#print(center)

#print(neighbors)

# 选择异号离子

if '+' in center.species\_string:

new\_neighbors = []

# 判断方式一：找到邻居内所有异号离子

# for i in neighbors:

# if "-" in i.species\_string:

# new\_neighbors.append(i)

# 判断方式二：找到排序后（从小到大排）的邻居内的离子，直到出现同号离子为止

for i in range(len(neighbors)):

if i is 0:

continue

if "-" in neighbors[i].species\_string:

new\_neighbors.append(neighbors[i].species\_string)

if "+" in neighbors[i].species\_string:

break

neighbors = new\_neighbors

if "-" in center.species\_string:

new\_neighbors = []

# for i in neighbors:

# if "+" in i.species\_string:

# new\_neighbors.append(i)

for i in range(len(neighbors)):

if i is 0:

continue

if "+" in neighbors[i].species\_string:

new\_neighbors.append(neighbors[i].species\_string)

if "-" in neighbors[i].species\_string:

break

neighbors = new\_neighbors

return len(neighbors)

# 自定义的Cif文件解析类

class CifParser\_new(CifParser):

"""

Parses a cif file

Args:

filename (str): Cif filename. bzipped or gzipped cifs are fine too.

occupancy\_tolerance (float): If total occupancy of a site is between 1

and occupancy\_tolerance, the occupancies will be scaled down to 1.

site\_tolerance (float): This tolerance is used to determine if two

sites are sitting in the same position, in which case they will be

combined to a single disordered site. Defaults to 1e-4.

"""

def \_\_init\_\_(self, filename, occupancy\_tolerance=1., site\_tolerance=1e-4):

self.\_occupancy\_tolerance = occupancy\_tolerance

self.\_site\_tolerance = site\_tolerance

if isinstance(filename, six.string\_types):

self.\_cif = CifFile.from\_file(filename)

else:

self.\_cif = CifFile.from\_string(filename.read())

# store if CIF contains features from non-core CIF dictionaries

# e.g. magCIF

self.feature\_flags = {}

self.errors = []

def is\_magcif():

"""

Checks to see if file appears to be a magCIF file (heuristic).

"""

# Doesn't seem to be a canonical way to test if file is magCIF or

# not, so instead check for magnetic symmetry datanames

prefixes = ['\_space\_group\_magn', '\_atom\_site\_moment',

'\_space\_group\_symop\_magn']

for d in self.\_cif.data.values():

for k in d.data.keys():

for prefix in prefixes:

if prefix in k:

return True

return False

self.feature\_flags['magcif'] = is\_magcif()

def is\_magcif\_incommensurate():

"""

Checks to see if file contains an incommensurate magnetic

structure (heuristic).

"""

# Doesn't seem to be a canonical way to test if magCIF file

# describes incommensurate strucure or not, so instead check

# for common datanames

if not self.feature\_flags["magcif"]:

return False

prefixes = ['\_cell\_modulation\_dimension', '\_cell\_wave\_vector']

for d in self.\_cif.data.values():

for k in d.data.keys():

for prefix in prefixes:

if prefix in k:

return True

return False

self.feature\_flags['magcif\_incommensurate'] = is\_magcif\_incommensurate()

for k in self.\_cif.data.keys():

# pass individual CifBlocks to \_sanitize\_data

self.\_cif.data[k] = self.\_sanitize\_data(self.\_cif.data[k])

@staticmethod

def from\_string(cif\_string, occupancy\_tolerance=1.):

"""

Creates a CifParser from a string.

Args:

cif\_string (str): String representation of a CIF.

occupancy\_tolerance (float): If total occupancy of a site is

between 1 and occupancy\_tolerance, the occupancies will be

scaled down to 1.

Returns:

CifParser

"""

stream = cStringIO(cif\_string)

return CifParser\_new(stream, occupancy\_tolerance)

def \_unique\_coords(self, coords\_in, magmoms\_in=None, lattice=None):

"""

Generate unique coordinates using coord and symmetry positions

and also their corresponding magnetic moments, if supplied.

"""

coords = []

coords\_num = []

if magmoms\_in:

magmoms = []

if len(magmoms\_in) != len(coords\_in):

raise ValueError

for tmp\_coord, tmp\_magmom in zip(coords\_in, magmoms\_in):

count = 0

for op in self.symmetry\_operations:

coord = op.operate(tmp\_coord)

coord = np.array([i - math.floor(i) for i in coord])

if isinstance(op, MagSymmOp):

# Up to this point, magmoms have been defined relative

# to crystal axis. Now convert to Cartesian and into

# a Magmom object.

magmom = Magmom.from\_moment\_relative\_to\_crystal\_axes(

op.operate\_magmom(tmp\_magmom),

lattice=lattice

)

else:

magmom = Magmom(tmp\_magmom)

if not in\_coord\_list\_pbc(coords, coord,

atol=self.\_site\_tolerance):

coords.append(coord)

magmoms.append(magmom)

count = count + 1

coords\_num.append(count)

return coords, magmoms, coords\_num

else:

for tmp\_coord in coords\_in:

count = 0

for op in self.symmetry\_operations:

coord = op.operate(tmp\_coord)

coord = np.array([i - math.floor(i) for i in coord])

if not in\_coord\_list\_pbc(coords, coord,

atol=self.\_site\_tolerance):

coords.append(coord)

count = count + 1

coords\_num.append(count)

return coords, [Magmom(0)] \* len(coords), coords\_num # return dummy magmoms

def \_get\_structure(self, data, primitive):

"""

Generate structure from part of the cif.

"""

def get\_num\_implicit\_hydrogens(sym):

num\_h = {"Wat": 2, "wat": 2, "O-H": 1}

return num\_h.get(sym[:3], 0)

lattice = self.get\_lattice(data)

# if magCIF, get magnetic symmetry moments and magmoms

# else standard CIF, and use empty magmom dict

if self.feature\_flags["magcif\_incommensurate"]:

raise NotImplementedError(

"Incommensurate structures not currently supported.")

elif self.feature\_flags["magcif"]:

self.symmetry\_operations = self.get\_magsymops(data)

magmoms = self.parse\_magmoms(data, lattice=lattice)

else:

self.symmetry\_operations = self.get\_symops(data)

magmoms = {}

oxi\_states = self.parse\_oxi\_states(data)

coord\_to\_species = OrderedDict()

coord\_to\_magmoms = OrderedDict()

def get\_matching\_coord(coord):

keys = list(coord\_to\_species.keys())

coords = np.array(keys)

for op in self.symmetry\_operations:

c = op.operate(coord)

inds = find\_in\_coord\_list\_pbc(coords, c,

atol=self.\_site\_tolerance)

# cant use if inds, because python is dumb and np.array([0]) evaluates

# to False

if len(inds):

return keys[inds[0]]

return False

label\_el\_dict = {}

for i in range(len(data["\_atom\_site\_label"])):

try:

# If site type symbol exists, use it. Otherwise, we use the

# label.

symbol = self.\_parse\_symbol(data["\_atom\_site\_type\_symbol"][i])

label = data["\_atom\_site\_label"][i]

num\_h = get\_num\_implicit\_hydrogens(

data["\_atom\_site\_type\_symbol"][i])

except KeyError:

symbol = self.\_parse\_symbol(data["\_atom\_site\_label"][i])

label = data["\_atom\_site\_label"][i]

num\_h = get\_num\_implicit\_hydrogens(data["\_atom\_site\_label"][i])

if not symbol:

continue

if oxi\_states is not None:

o\_s = oxi\_states.get(symbol, 0)

# use \_atom\_site\_type\_symbol if possible for oxidation state

if "\_atom\_site\_type\_symbol" in data.data.keys():

oxi\_symbol = data["\_atom\_site\_type\_symbol"][i]

o\_s = oxi\_states.get(oxi\_symbol, o\_s)

try:

el = Specie(symbol, o\_s)

except:

el = DummySpecie(symbol, o\_s)

else:

el = get\_el\_sp(symbol)

x = str2float(data["\_atom\_site\_fract\_x"][i])

y = str2float(data["\_atom\_site\_fract\_y"][i])

z = str2float(data["\_atom\_site\_fract\_z"][i])

magmom = magmoms.get(data["\_atom\_site\_label"][i],

np.array([0, 0, 0]))

try:

occu = str2float(data["\_atom\_site\_occupancy"][i])

except (KeyError, ValueError):

occu = 1

if occu > 0:

coord = (x, y, z)

match = get\_matching\_coord(coord)

comp\_d = {el: occu}

if num\_h > 0:

comp\_d["H"] = num\_h

comp = Composition(comp\_d)

if not match:

coord\_to\_species[coord] = comp

coord\_to\_magmoms[coord] = magmom

else:

coord\_to\_species[match] += comp

# disordered magnetic not currently supported

coord\_to\_magmoms[match] = None

label\_el\_dict[coord] = label

sum\_occu = [sum(c.values()) for c in coord\_to\_species.values()

if not set(c.elements) == {Element("O"), Element("H")}]

if any([o > 1 for o in sum\_occu]):

msg = "Some occupancies (%s) sum to > 1! If they are within " \

"the tolerance, they will be rescaled." % str(sum\_occu)

warnings.warn(msg)

self.errors.append(msg)

allspecies = []

allcoords = []

allmagmoms = []

allhydrogens = []

alllabels = []

# check to see if magCIF file is disordered

if self.feature\_flags["magcif"]:

for k, v in coord\_to\_magmoms.items():

if v is None:

# Proposed solution to this is to instead store magnetic

# moments as Specie 'spin' property, instead of site

# property, but this introduces ambiguities for end user

# (such as unintended use of `spin` and Specie will have

# fictious oxidation state).

raise NotImplementedError(

'Disordered magnetic structures not currently supported.')

if coord\_to\_species.items():

for comp, group in groupby(

sorted(list(coord\_to\_species.items()), key=lambda x: x[1]),

key=lambda x: x[1]):

tmp\_coords = [site[0] for site in group]

#print(tmp\_coords)

labels = []

for i in tmp\_coords:

labels.append(label\_el\_dict[i])

#print(labels)

tmp\_magmom = [coord\_to\_magmoms[tmp\_coord] for tmp\_coord in

tmp\_coords]

if self.feature\_flags["magcif"]:

coords, magmoms, coords\_num = self.\_unique\_coords(tmp\_coords,

magmoms\_in=tmp\_magmom,

lattice=lattice)

else:

coords, magmoms, coords\_num = self.\_unique\_coords(tmp\_coords)

if set(comp.elements) == {Element("O"), Element("H")}:

# O with implicit hydrogens

im\_h = comp["H"]

species = Composition({"O": comp["O"]})

else:

im\_h = 0

species = comp

allhydrogens.extend(len(coords) \* [im\_h])

allcoords.extend(coords)

allspecies.extend(len(coords) \* [species])

allmagmoms.extend(magmoms)

for i in range(len(coords\_num)):

alllabels.extend(coords\_num[i] \* [labels[i]])

# rescale occupancies if necessary

for i, species in enumerate(allspecies):

totaloccu = sum(species.values())

if 1 < totaloccu <= self.\_occupancy\_tolerance:

allspecies[i] = species / totaloccu

if allspecies and len(allspecies) == len(allcoords) \

and len(allspecies) == len(allmagmoms):

site\_properties = dict()

if any(allhydrogens):

assert len(allhydrogens) == len(allcoords)

site\_properties["implicit\_hydrogens"] = allhydrogens

if self.feature\_flags["magcif"]:

site\_properties["magmom"] = allmagmoms

if len(site\_properties) == 0:

site\_properties = None

struct = Structure(lattice, allspecies, allcoords,

site\_properties=site\_properties)

#struct = struct.get\_sorted\_structure()

if primitive and self.feature\_flags['magcif']:

struct = struct.get\_primitive\_structure(use\_site\_props=True)

elif primitive:

struct = struct.get\_primitive\_structure()

struct = struct.get\_reduced\_structure()

struct.add\_site\_property("\_atom\_site\_label", alllabels)

return struct

#自定义半径数据结构

class Radius():

def \_\_init\_\_(self, label, element, coord\_num, radius):

self.\_label = label

self.\_element = element

self.\_coord\_num = coord\_num

self.\_radius = radius

#列表labels, elements, coord\_nums, radii必须具有相同的大小

@staticmethod

def get\_radii\_list(labels, elements, coord\_nums, radii):

radii\_list = []

if len(labels) != len(elements) != len(coord\_nums) != len(radii):

raise ValueError("labels, elements, coord\_nums, radii must be have same length!")

for i in range(len(labels)):

radii\_list.append(Radius(labels[i], elements[i], coord\_nums[i], radii[i]).as\_dict())

return radii\_list

def as\_dict(self):

"""

Dict representation of Radius.

Returns:

JSON serializable dict representation.

"""

d = {"label": self.\_label,

"element": self.\_element,

"coord\_num": self.\_coord\_num,

"radius": self.\_radius}

# # 有序字典

# d = OrderedDict()

# d["label"] = self.\_label

# d["element"] = self.\_element

# d["coord\_num"] = self.\_coord\_num

# d["radius"] = self.\_radius

return d

def get\_ionic\_radii(filename):

"""

Computes ionic radii of elements for all sites in the structure.

If valence is zero, atomic radius is used.

"""

radii = []

cn = []

labels = []

radii2 = []

cn2 = []

el2 = []

labels2 = []

vnn = VoronoiNN\_self(cutoff = 10.0)

with zopen(filename, "rt") as f:

input\_string = f.read()

parser = CifParser\_new.from\_string(input\_string)

stru = parser.get\_structures(primitive=False)[0]

#print(stru)

def nearest\_key(sorted\_vals, key):

i = bisect\_left(sorted\_vals, key)

if i == len(sorted\_vals):

return sorted\_vals[-1]

if i == 0:

return sorted\_vals[0]

before = sorted\_vals[i-1]

after = sorted\_vals[i]

if after-key < key-before:

return after

else:

return before

for i in range(len(stru.sites)):

site = stru.sites[i]

#print(site)

if isinstance(site.specie, Element):

radius = site.specie.atomic\_radius

# Handle elements with no atomic\_radius

# by using calculated values instead.

if radius is None:

radius = site.specie.atomic\_radius\_calculated

if radius is None:

raise ValueError(

"cannot assign radius to element {}".format(

site.specie))

radii.append(radius)

continue

el = site.specie.symbol

oxi\_state = int(round(site.specie.oxi\_state))

coord\_no = int(round(vnn.get\_cn(stru, i)))

label = site.\_atom\_site\_label

try:

tab\_oxi\_states = sorted(map(int, \_ion\_radii[el].keys()))

oxi\_state = nearest\_key(tab\_oxi\_states, oxi\_state)

radius = \_ion\_radii[el][str(oxi\_state)][str(coord\_no)]

except KeyError:

if vnn.get\_cn(stru, i)-coord\_no > 0:

new\_coord\_no = coord\_no + 1

else:

new\_coord\_no = coord\_no - 1

try:

radius = \_ion\_radii[el][str(oxi\_state)][str(new\_coord\_no)]

coord\_no = new\_coord\_no

except:

tab\_coords = sorted(map(int, \_ion\_radii[el][str(oxi\_state)].keys()))

new\_coord\_no = nearest\_key(tab\_coords, coord\_no)

i = 0

for val in tab\_coords:

if val > coord\_no:

break

i = i + 1

if i == len(tab\_coords):

key = str(tab\_coords[-1])

radius = \_ion\_radii[el][str(oxi\_state)][key]

elif i == 0:

key = str(tab\_coords[0])

radius = \_ion\_radii[el][str(oxi\_state)][key]

else:

key = str(tab\_coords[i-1])

radius1 = \_ion\_radii[el][str(oxi\_state)][key]

key = str(tab\_coords[i])

radius2 = \_ion\_radii[el][str(oxi\_state)][key]

radius = (radius1+radius2)/2

#implement complex checks later

# #保存所有位置的配位数、半径和labels

# cn.append(coord\_no)

# radii.append(radius)

# labels.append(label)

if label in labels2:

continue

labels2.append(label)

radii2.append(radius)

cn2.append(coord\_no)

el2.append(site.species\_string)

radii\_dict = dict(zip(el2,radii2))

return radii\_dict

#以自定的数据结构Radius列表形式返回

# return Radius.get\_radii\_list(labels2, el2, cn2, radii2)

# #所有位置的配位数、半径

# el = [site.species\_string for site in stru.sites]

# cn\_list = list(zip(el, cn))

# radii\_list = list(zip(el, radii))

# radii\_cn\_list = list(zip(el, cn, radii))

# label\_cn\_radii\_list = list(zip(labels, cn, radii))

#按label计算出配位数、半径

# cn\_list = list(zip(el2, cn2))

# radii\_list = list(zip(el2, radii2))

# radii\_cn\_list = list(zip(el2, cn2, radii2))

# label\_cn\_radii\_list = list(zip(labels2, el2, cn2, radii2))

# return radii\_list, cn\_list, radii\_cn\_list,label\_cn\_radii\_list