CAVD用户手册

叶安江

0.3版本

20180816

目录

[一、 CAVD简介 3](#_Toc519135940)

[二、 CAVD的特点 3](#_Toc519135941)

[三、 CAVD的主要技术 4](#_Toc519135942)

[四、 CAVD的主要功能 4](#_Toc519135943)

[五、 CAVD的目录结构 4](#_Toc519135944)

[六、 CAVD的安装方法 5](#_Toc519135945)

[七、 CAVD的计算原理 6](#_Toc519135946)

[八、 CAVD的计算源数据说明 8](#_Toc519135947)

[九、 CAVD的参数定义 8](#_Toc519135948)

[十、 CAVD的代码架构 11](#_Toc519135949)

[十一、 CAVD的文件格式说明 12](#_Toc519135950)

[十二、 CAVD的接口说明 15](#_Toc519135951)

[zeo.BIComputation 16](#_Toc519135952)

[zeo.Connection 18](#_Toc519135953)

[zeo.BIComputation\_batch 19](#_Toc519135954)

[zeo.Connection\_batch 20](#_Toc519135955)

[zeo.Connection\_batch 21](#_Toc519135956)

[zeo. com 22](#_Toc519135957)

[十三、 CAVD的接口使用案例 23](#_Toc519135958)

## CAVD简介

CAVD(Crystal structure Analysis by Voronoi Decomposition)，是基于Voronoi分解方法的晶体结构分析库，包含C++和Python两种版本库，可以方便的用于晶体结构的几何分析及其内部空隙空间的拓扑分析，具体可描述为：利用CAVD可获取晶体结构的几何属性，可计算快离子导体中迁移离子瓶颈和间隙的分布与尺寸，进而可寻找迁移离子路径等信息。CAVD不仅可以用于分析单个结构，而且可为大型数据库进行高通量分析，为分析和了解固态电解质化合物的物理特性提供有力的支持。

## CAVD的特点

1. 操作简单，仅需给定结构文件即可进行计算。
2. 计算高效，计算单一结构所需耗时在秒级。
3. 兼容性强，提供C++和Python两种接口，方便调用。
4. 方便可视化，数据结构合理，方便后续的可视化分析。
5. 提供多种呈现数据的方式。

## CAVD的主要技术

1. 通过Voronoi分解方法进行晶体结构的几何属性和空隙拓扑分析，方便、高效且准确。Voronoi是1908年由俄国科学家Georgy Fedosievych Voronyi根据笛卡尔用凸域分割空间的思想，建立出的一套新的空间分割算法。他定义并研究了n维普通Voronoi图（the general n-dimensional case），Voronoi图也由此得名。Voronoi 分解法即是使用Voronoi空间分割算法划分空间的一种方法，利用该方法可将空间完全划分为若干个（依据离散点的数目而定）Voronoi Cell，且相邻的Voronoi Cell共边（或面），且该边（或面）为两个相邻Voronoi Cell的垂直平分线（或面），因此可应用到多个领域，并且都取得了相当多的成果。
2. 基于开源的Zeo++库进行CAVD的开发。Zeo++是用于分析晶体多孔材料的软件包，可用于对材料内部空隙空间进行表征和分析。zeo++利用开源库voro++进行底层的Voronoi图计算，Voro++是一款计算Voronoi图的开源软件库，它在为一个粒子系统构建Voronoi图时，采取基于Cell的策略，并可以考虑非正交周期性边界条件，这使得统计与cell相关的信息（如体积， 面，Voronoi顶点等）变得方便。同时，Voro++和Zeo++都是用面向对象的C++语言编写的，能够提供显著高效的运行速度。
3. 利用Cython语言将C++源代码与Python拓展模块相结合，提供了方便高效的python接口。CAVD库的底层实现为面向对象的C++代码，通过Cython语言包装C++，将其封装成Python语言可调用的接口模块，从而实现了C++与Python的结合，使得程序代码在运行速度与使用方便两种不同的考量方式下找到了一个平衡，也使得程序的代码便于维护与扩展。
4. CAVD主要是在Zeo++代码的基础上做了以下改进：

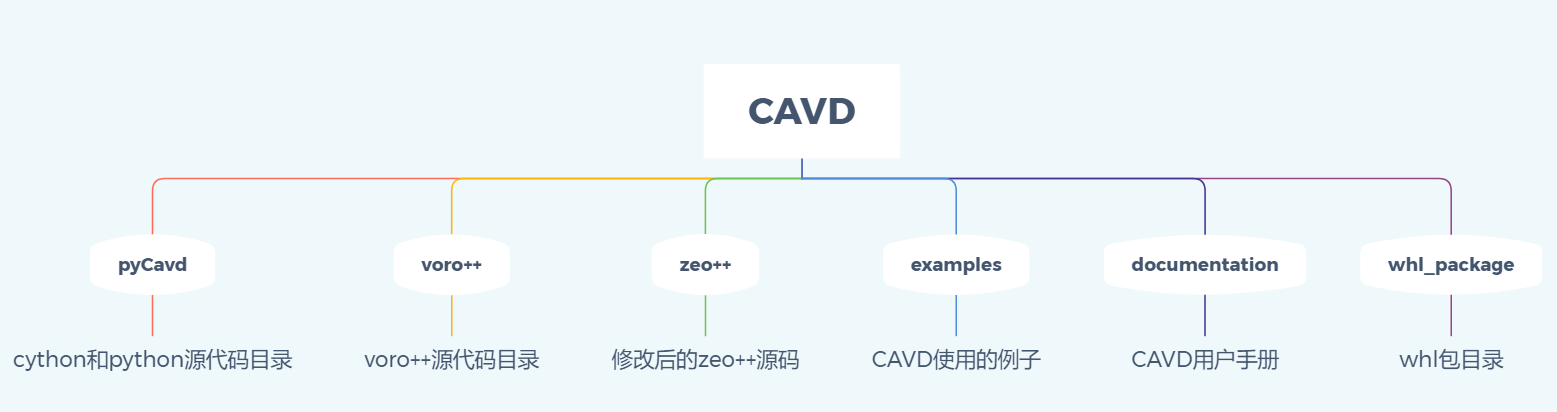
* 通过对晶体结构中离子配位数的精确计算以及引入准确度高、认可范围广的香农有效离子半径表，可计算出晶体结构中离子的有效半径。在准确的有效离子半径的基础上，进而为其后进行的微观结构几何分析提供了结果准确性的保障。
* 用python代码实现了骨架离子堆积方式的确定程序，增加了程序功能以及未来的可扩展性。
* 扩展了文件的输出格式，增加了.vasp等格式文件，方便后续的可视化操作。
* 增加了输入文件的处理，可自动去除结构文件中某一原子的信息，从而使程序能够更方便的应用与固体电解质的分析。
* 为zeo++代码中C++实现的某些功能模块，编写了python接口，使其方便的编译成python拓展模块，方便调用。
* 为zeo++代码在Windows平台上的适用进行了一定的代码修改，使其完美适配Windows平台。

## CAVD的主要功能

1. 晶体结构中间隙与瓶颈的分布与尺寸的计算。
2. 晶体结构中迁移离子通道的计算。
3. 晶体结构结构导通性计算。
4. 晶体结构中迁移离子可访问区域表面积与体积的计算。

## CAVD的目录结构

下载的CAVD为安装包文件，解压之后的目录结构如下：



CAVD为安装包文件根目录，该目录下包含6个子目录。

pyCavd目录中为Cython和Python源代码，其中Cython源代码为zeo++目录下源代码的封装，Python源代码为实现新功能的代码。

voro++目录中主要为C++代码，是voro++源代码。

zeo++目录中主要为C++代码，是根据晶体结构几何分析的需求改进的zeo++代码。

examples目录中是利用CAVD进行计算的一些例子。

documentation目录中是有关CAVD的文档信息，包括用户手册等。

whl\_packages目录中是CAVD在不同环境下的安装包。

## CAVD的安装方法

CAVD提供Linux和Windows平台下多种版本的安装包，需满足的依赖环境为：

* python3.5以上
* Cython 0.28以上
* pymatgen库

CAVD安装包中的“whl\_packages”目录中包含中一些不同环境下的.whl安装包，用户可简单使用.whl包进行安装。当“whl\_packages”目录中不满足当前需求时，还可以直接从源码进行编译安装以适应不同的环境需要。

现对安装方法做一个简单的介绍（在windows下使用利用whl包进行安装示范，linux平台下从源码进行安装示例）：

1. windows系统下从.whl包进行安装（需提前安装依赖环境）：

**zeo-0.1-cp35-cp35m-win\_amd64.whl** windows X64 Python 3.5

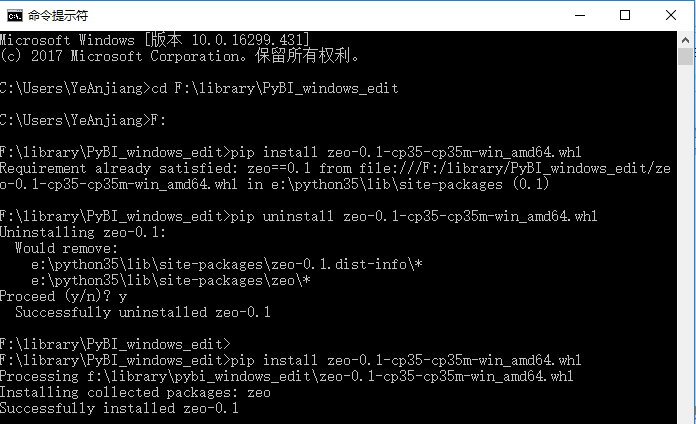
**zeo-0.1-cp36-cp36m-win\_amd64.whl** windows X64 Python 3.6

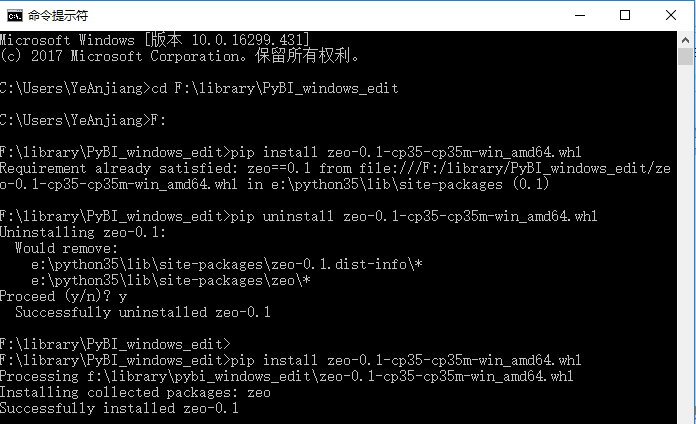
**zeo-0.1-cp35-cp35m-linux\_x86\_64.whl**  linux X64 Python 3.5

1. 下载对应版本的whl包，并放置到适当目录。
2. 在cmd中将目录转到上一步中的目录下，运行如下命令（如图所示）：

**pip install zeo-0.1-cp35-cp35m-win\_amd64.whl**

当出现Successfully installed zeo-0.1 表示安装结束。





1. 验证是否安装成功。在cmd下输入

**python**

**import zeo**

若不报错，说明安装成功。

1. Linux系统下从源码进行安装（需提前安装依赖环境）：
2. 下载CAVD安装包。
3. 在命令行中将目录转到CAVD目录，并输入如下命令预先安装voro++库。

**cd voro++**

**make**

1. 在命令行中将目录转到zeo++目录，之后将修改后的zeo++编译成动态链接库，输入的命令如下：

**cd ..**

**cd zeo++**

**make dylib**

1. 在命令行中将目录转到PyCavd目录，并输入

**python setup\_alt install**

当出现Successfully installed zeo-0.1 表示安装结束。

1. 验证是否安装成功。输入如下命令：

**python**

**import zeo**

若不报错，说明安装成功。

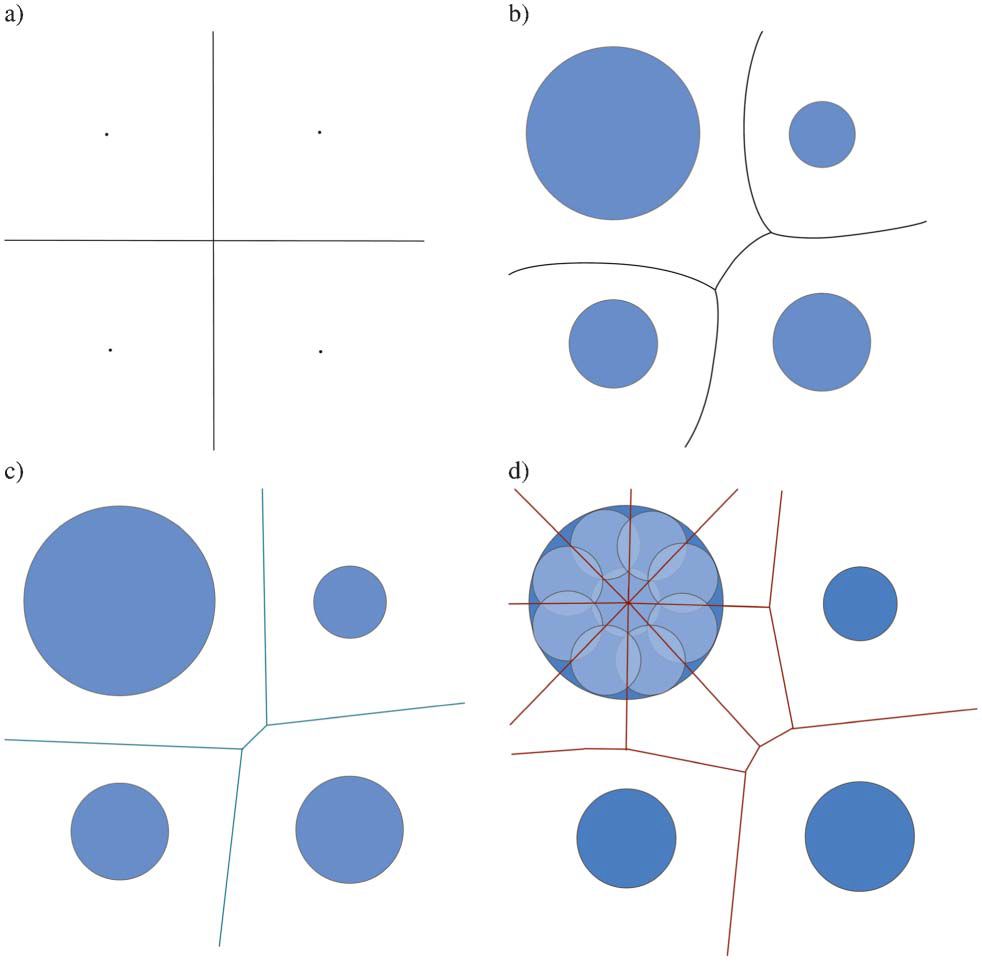
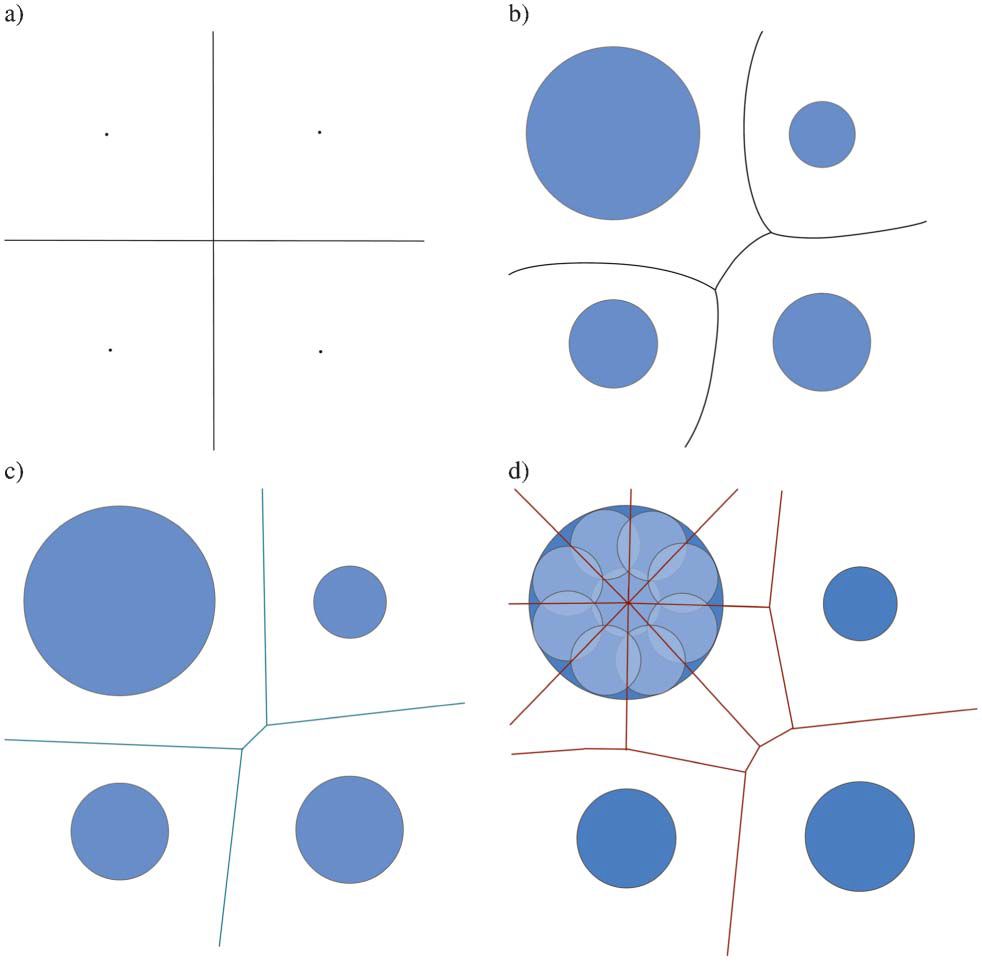
安装完成之后，即可在Python程序中轻松使用import命令导入所需使用的接口来完成相应的计算任务。

## CAVD的计算原理

晶体中各离子(原子)间的相互结合，可以看作是球体的堆积。对于离子晶体来说，阴离子做紧密堆积，阳离子填充在其空隙之中。从几何的角度分析晶体结构中迁移离子传导过程，首要任务是要找到迁移离子间隙与瓶颈位置，因此找到一种表征晶体结构中空隙空间的方法尤为重要。表征这种空隙空间的比较常见的方法是Voronoi 网络映射，即将空隙空间映射成一个Voronoi网络。

完成这种映射的方法称作Voronoi 分解方法。所谓Voronoi分解，即是将晶胞中的空间进行划分成多个多面体（cell），每个多面体由顶点和边组成。这些多面体相互邻居且将晶胞空间完美剖分，同时依据Voronoi的定义可知，每个离子(或原子)都被包含在一个多面体内，且在这个多面体内距离中心离子（原子）的距离小于其他任何离子（或原子）的距离，每个多面体的顶点处为空间中空隙较大处，迁移离子的瓶颈位于多面体的边上。CAVD中所用Voronoi多面体公式与定义的详细说明如下：

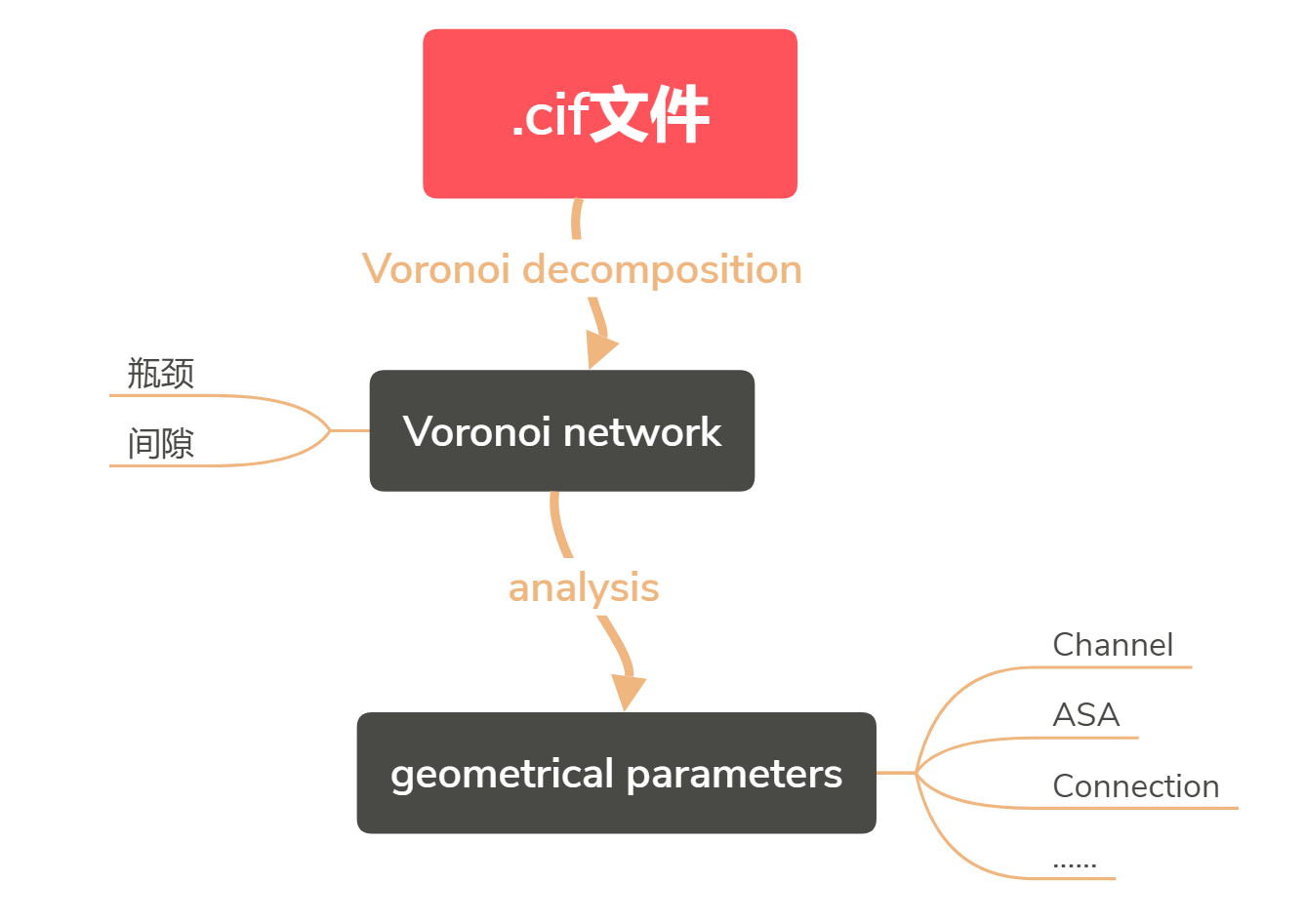
表示点x和点y之间的距离。每个多面体的面为相邻离子（或原子）形成的多面体的边界，由定义。在此种定义下，保证了多面体边界为非凹的，且与实际分布相似，做到了计算效率与准确性的平衡。如图



上图右侧为间隙空间的真实分布，由公式决定。上图左侧为CAVD所运用的公式划分的空间。

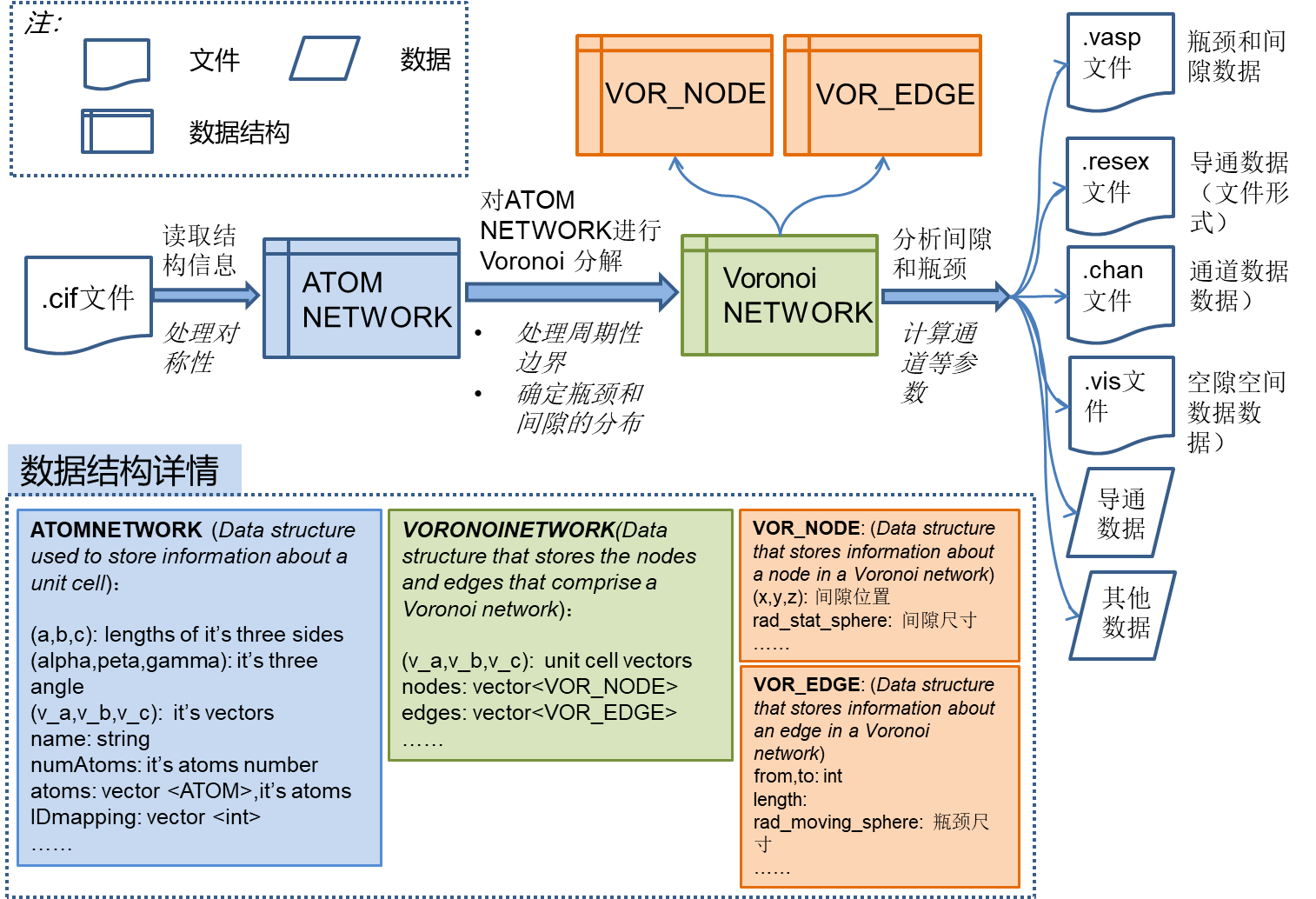
CAVD则是利用Voronoi分解方法，将晶体结构中的空隙空间映射成一个Voronoi网络。Voronoi网络是一组边（edge）和顶点（node）的集合，对于每个边和顶点会保存其与最近邻离子的距离以及产生该最短距离的位置，即为瓶颈和间隙的尺寸及位置。对于这样的Voronoi 网络利用路径搜索算法（Dijkstra算法），可得到迁移路径以及结构的连通性信息。

程序计算原理如下：



1. 读入结构文件
2. 利用空间群信息得到一个晶胞内的所有等效点系
3. 去除迁移离子
4. 运用Voronoi分解方法得到空隙空间的网络映射（Voronoi network）
5. 对Voronoi network进行分析，得到离子通道，迁移离子可访问区域等几何信息。

CAVD软件的计算流程如下图所示：



## CAVD涉及的源数据说明

1. 离子半径表

离子半径表选用香农1976年总结的有效离子半径表（Revised effective ionic radii and systematic studies of interatomic distances in halides and chalcogenides. R. D. Shannon Acta Cryst. (1976) A32, 751-767.）

## CAVD的参数定义

1. 瓶颈尺寸定义：

穿过瓶颈区时迁移离子到该区顶点离子表面的距离的最小值。瓶颈尺寸也即Voronoi 多面体边的最邻近距离，瓶颈位置则是边上与最短距离对应的点。公式推导的数学抽象如下：

已知： 半径为，A，B， , ,，为、、到线段AB距离取最小值时线段AB上对应那点，求到、、表面距离的最小值R和的坐标。

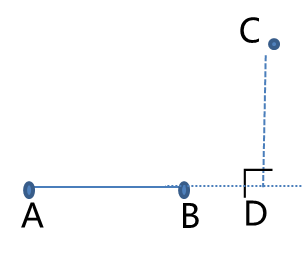
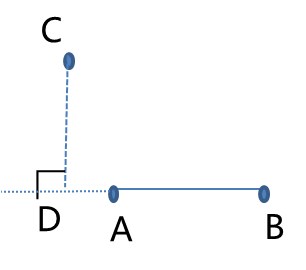
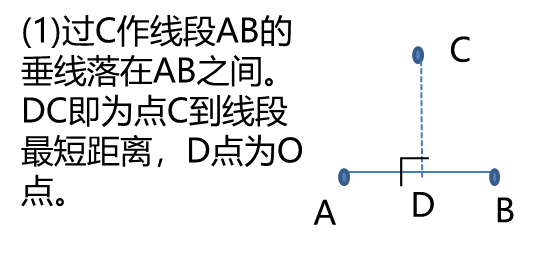
公式推导过程：

点C到线段AB的最短距离，有(1),(2),(3)三种情况：

(1)过C作线段AB的垂线落在AB之间。DC即为点C到线段最短距离，D点为O点。

(2)过C作线段AB的垂线落在AB延长线A这一侧之间。则C到线段AB最短距离为CA长度，A点为O点。

(3)过C作线段AB的垂线落在AB延长线B这一侧之间。则C到线段AB最短距离为CB长度，B点为O点。



分析得出，可用向量与的投影来判断上述三种情况，并用点积参与运算，计算出O的位置，并进一步求出点C到线段的最短距离。

对于向量与的点积 (为向量与的夹角)，表示向量与向量在向量上投影的乘积。

故向量在向量上投影为：

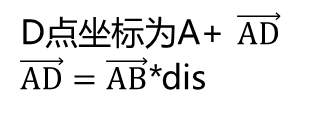
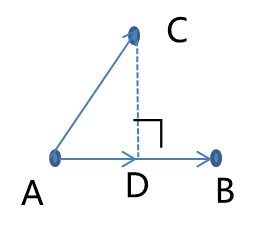
= ，

则dis为在向量上投影与向量的比值。

若，则对应(2)，O的坐标为A点坐标，最短距离为CA长度；

若，则对应(3)，O的坐标为B点最表，最短距离为CB长度；

若，则对应(1)，O的坐标为垂点D的坐标，最短距离为CD长度，如图。



综合上述三种情况，假设A点坐标为，对dis采取如下处理：

则O点坐标可统一表示成, 最短距离为CO长度。

根据以上原理可分别求出、、到线段AB的最短距离，再减去半径即得所需的值。

1. 间隙定义

距离形成配位多面体的骨架离子表面的最小距离。间隙位置也即Voronoi 多面体顶点中心位置，间隙尺寸即为多面体顶点位置到到最邻近离子（或原子）距离。公式推导的数学抽象如下：

已知：A，B , C , D以及对应的半径，，，求间隙及间隙大小d。



公式推导过程：

O点坐标为，又因为O是垂直平分面额交点，故有：

OA=OB=OC=OD

则O为四面体外接球球心。

记，令

则。

O到球A表面的距离为=

同理，可得出O到球B、球C、球D表面的距离、、。

则d=min{。

1. 通道定义

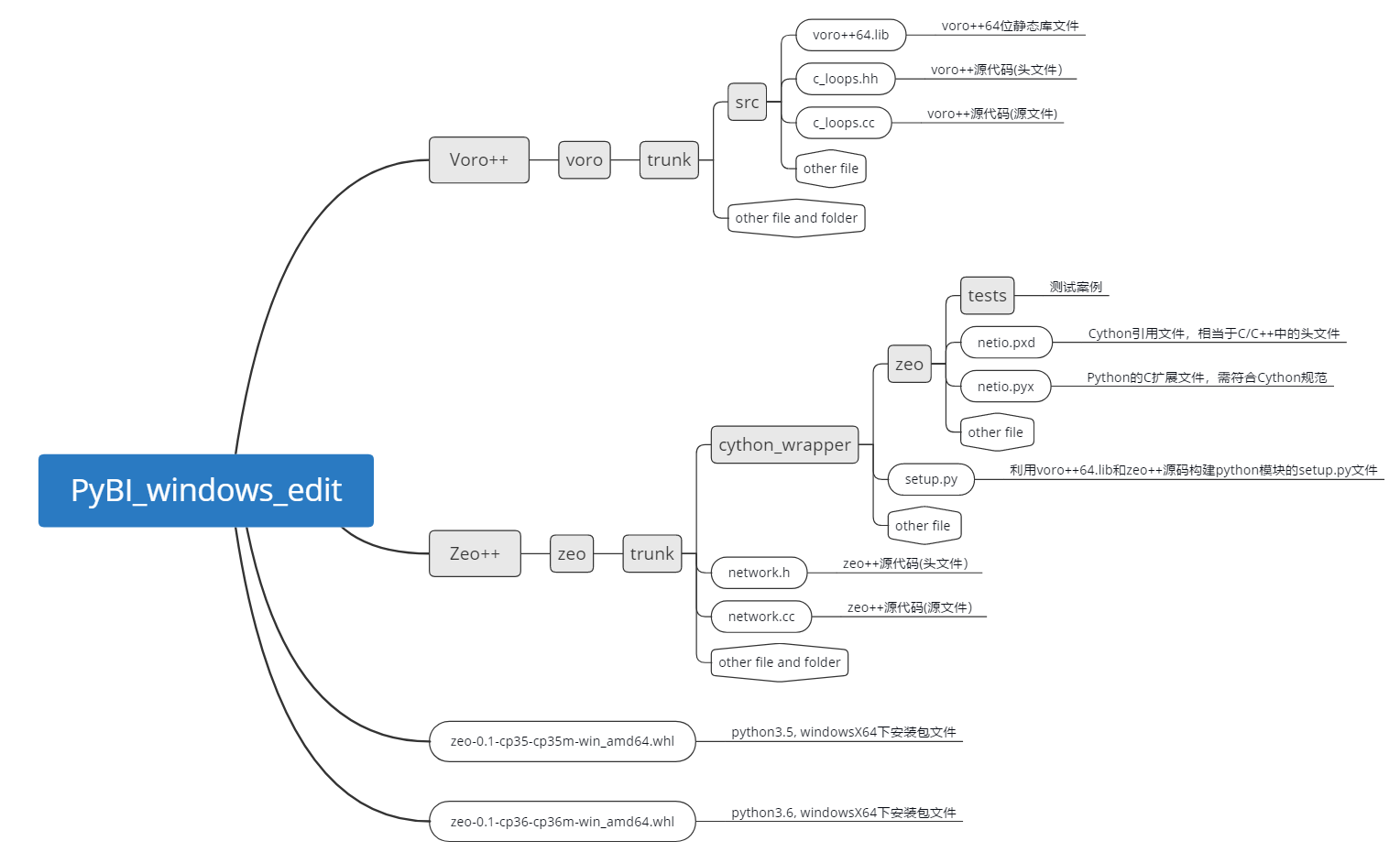
对于表征空隙空间的Voronoi网络，利用一定大小的探针去探寻该网络图，与每个顶点和边上所分配的间隙、瓶颈尺寸进行对比，若探针的尺寸小于该尺寸，表示该结点和边可被访问，所有可访问的结点和边组成通道。

通道确定算法如下：

1. 计算给定结构一个晶胞内的周期性Voronoi 网络(periodic Voronoi network)
2. 去除Voronoi网络中与最近邻原子表面的距离小于探针半径的所有节点（node）和边（edge）。
3. 选择一个未被检查过（visited）的节点作为起点，并将其周期性位移矢量（PDV）记录为（000）。 用从该节点出发的边（edge），将所有与该边直接相连的节点的ID和PDV放置在栈里。 当栈里还有节点时，将最顶端的节点出栈，并执行以下分析：
   1. 如果节点未被检查过，则记录其ID和PDV，并将其所有直接相连的节点的ID和PDV添加到堆栈。
   2. 如果被不同的PDV访问，则所有与该PDV对应的节点都可访问。一旦堆栈为空，如果节点未被标记为可访问，则它们都不可访问。
4. 重复步骤3，直到所有节点都被检查完毕。
5. 迁移离子可访问区域面积（ASA）定义
6. 导通性定义

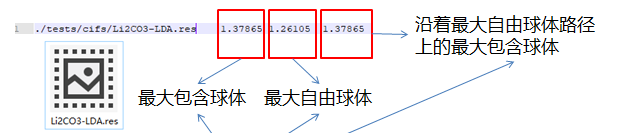
最大自由球体的定义为：最大能自由通过当前Voronoi网络边和结点的半径值，也即是所有间隙和瓶颈尺寸的最小值。对于晶体结构来说，分别计算在a、b、c方向上的最大自由球体，即可判断出一维、二维、三维导通性。

## CAVD的代码架构

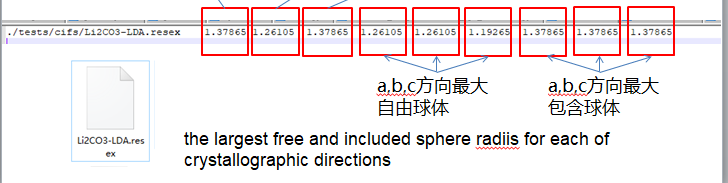


## CAVD的文件格式说明

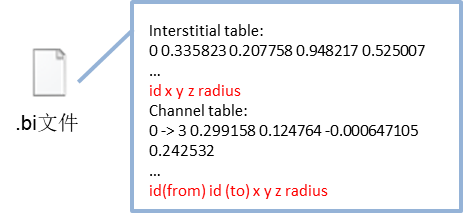
1. \*.res文件：连通性文件



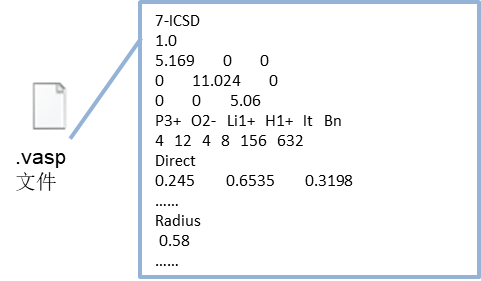
1. \*.resex文件：连通性文件



1. \*.bi文件：瓶颈、间隙尺寸和位置信息

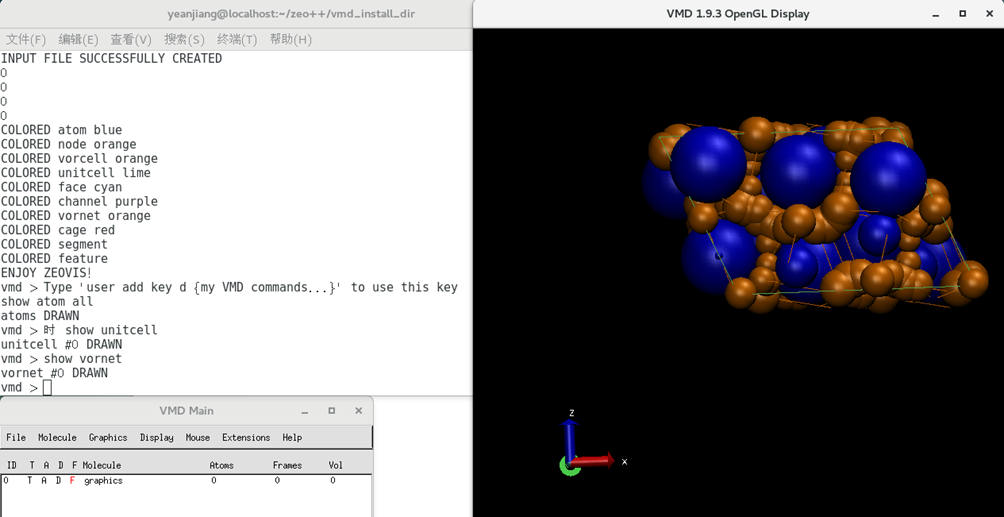


1. \*.vasp文件：瓶颈、间隙尺寸和位置信息（可视化数据；其中\*\_selected.vasp为按尺寸筛选后的信息）



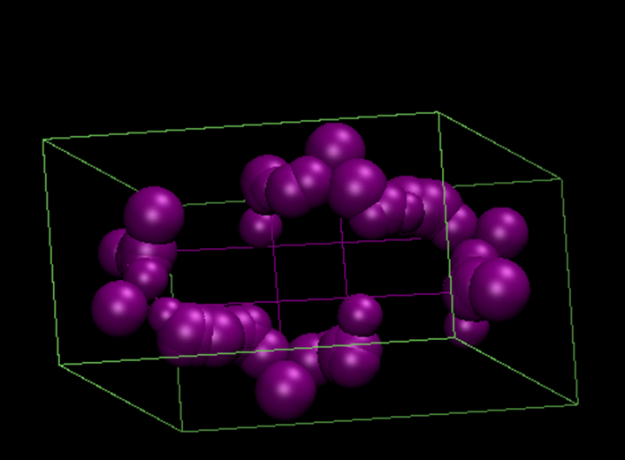
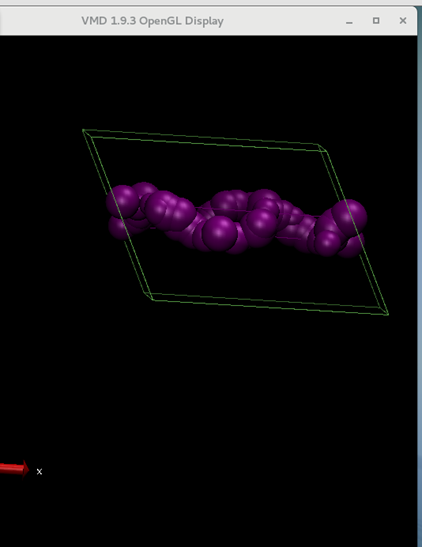
1. \*.zvis文件：Voronoi network数据文件（用于VMD的可视化数据；后续可自定义数据结构）

在VMD中显示的icsd\_16713.cif的空隙空间与结构如下所示：



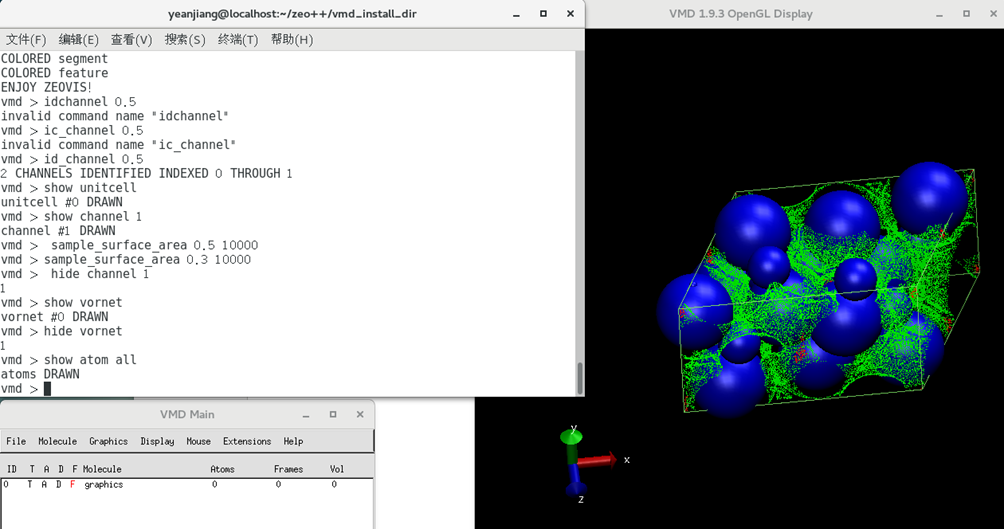
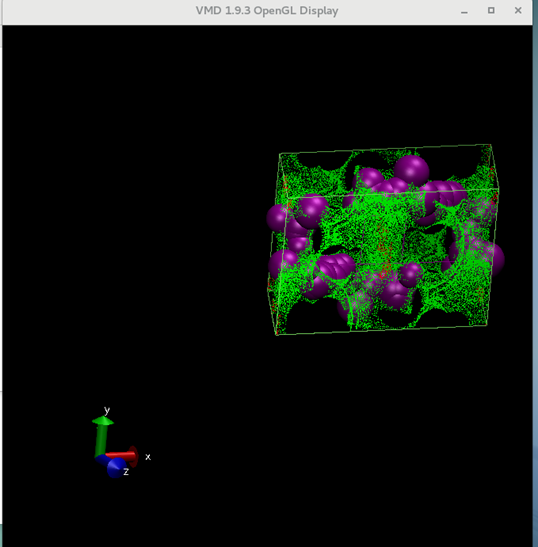
1. \*.zchan文件：通道信息文件（用于VMD的可视化数据；后续可自定义数据结构）

在VMD中显示icsd\_16713.cif在0.5A探针下的通道（2条）如下：

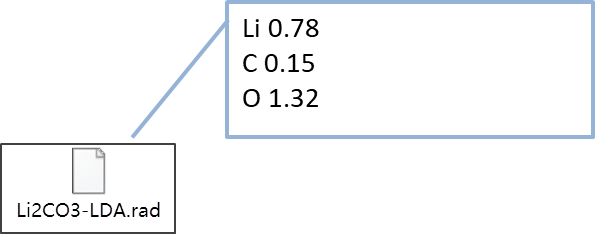


1. \*.zsa文件：可访问区域数据文件（用于VMD的可视化数据；后续可自定义数据结构）

在VMD下显示icsd\_16713.cif在0.5A，每个原子采样点为10000的探针采样所显示的ASA如下



1. \*.rad文件：自定义的输入半径文件。格式如下：



## CAVD的接口说明

瓶颈和间隙计算程序提供十个函数接口，分别是：

zeo. EffectiveRadCom()：计算输入的.cif文件所表示的结构中各种离子的半径。

zeo.BIComputation()：计算输入的.cif文件所表示的结构中瓶颈与间隙的分布及尺寸，并保存在.vasp文件和.bi文件中。

zeo.ConnValCom()：计算输入的.cif文件所表示的结构中的最大自由球体半径，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

zeo. ConnValListCom ()：计算输入的.cif文件所表示的结构中的连通性状态列表，存放1D，2D，3D连通半径，这些元素组成一个列表

zeo. ConnStatusCom()：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

zeo. ConnStatus()：计算输入的.cif文件所表示的结构中的迁移离子通道

zeo.ChannelCom：读取指定目录下所有.cif文件，根据输入参数，计算其瓶颈和间隙的位置与尺寸。

zeo. ASACom ()：计算输入的.cif文件所表示的结构中的可访问表面积。

zeo. VoidNetCom()：计算输入的.cif文件所表示的结构中的空隙网络。

zeo.AllCom()：计算输入的.cif文件所表示的结构中瓶颈和间隙分布及尺寸，导通半径值列表，导通属性，迁移离子通道，可访问表面积和间隙空间。

详细介绍如下：

### zeo.BIComputation

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

**功能：**从输入的.cif文件中读取结构信息，根据输入参数，计算出该结构中瓶颈与间隙的位置以及尺寸，并保存在.vasp文件和.bi文件中。

**参数：**

filename：输入的cif文件名，此文件中的结构信息是后续瓶颈与间隙计算的基础。

migrant：迁移离子。在指定某离子为迁移离子的情况下，程序会先去除cif文件中所有的迁移离子的位置信息，并生成一个新的cif文件（中间文件）。之后程序会读入这个新的cif文件，获取其中的结构信息进行后续的瓶颈与间隙计算。缺省值为None，程序直接读取输入的cif文件中的结构信息进行后续的瓶颈与间隙计算。

rad\_flag：半径标志位，表示是否使用半径进行瓶颈和间隙计算。rad\_flag = True，表明使用半径进行计算；rad\_flag = False，表明不使用半径进行计算。

rad\_file：在rad\_flag为True时，程序会判定是否给定输入的半径文件（.rad文件）。如果指定了半径文件则使用半径文件中的半径信息参与后续瓶颈和间隙计算，否则使用程序默认半径进行后续计算。半径文件需与输入结构文件具有相同的前缀名，后缀为“.rad”；文件中的内容格式为：每行指定一个元素的半径。rad\_file的缺省值为None。

rad\_store\_in\_vasp：指示是否在输出的vasp文件中保存半径信息。如果rad\_store\_in\_vasp = True，则保存半径信息；如果rad\_store\_in\_vasp = False，则不保存半径信息。

minRad：用于筛选计算得到的瓶颈与间隙。当瓶颈和间隙的尺寸小于minRad时，则不保存。但当设置minRad=0时，保存所有瓶颈和间隙信息。

**输入：**[.cif文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.cif)，[.rad文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.rad)。

**输出：**[.bi文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA.bi)（保存瓶颈和间隙信息），[.vasp文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA.vasp)（保存结构信息和计算得到的瓶颈和间隙信息），[\*\_remove[ION].cif](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA_removeLi.cif)文件（[ION]为迁移离子，去除迁移离子后生成的中间文件）。

**示例：**计算当前目录下Li2CO3-LDA.cif结构中，Li离子迁移过程中的瓶颈和间隙，考虑半径并指定自定义的半径文件，并将保留半径大于0.5的瓶颈和间隙。

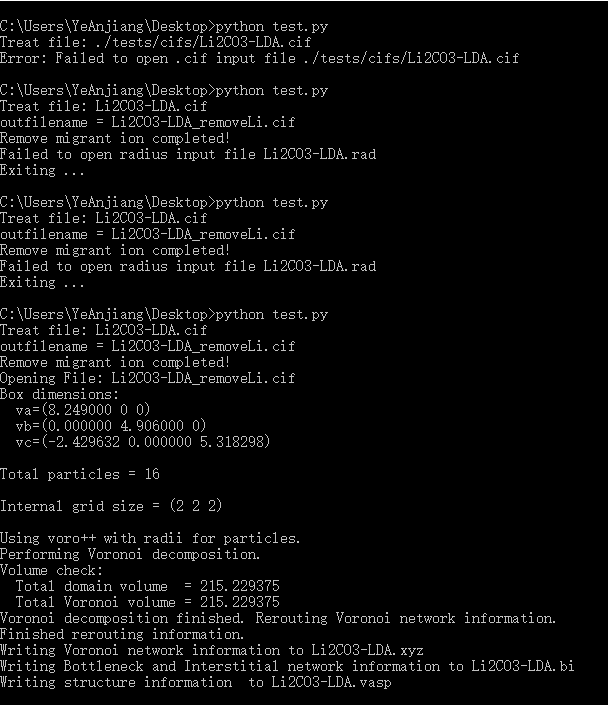
代码如下：

# test.py

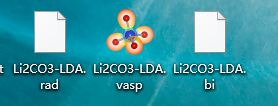
import zeo

zeo.BIComputation(“Li2CO3-LDA.cif”, “Li”, True, ”Li2CO3-LDA.rad”, True, 0.5)

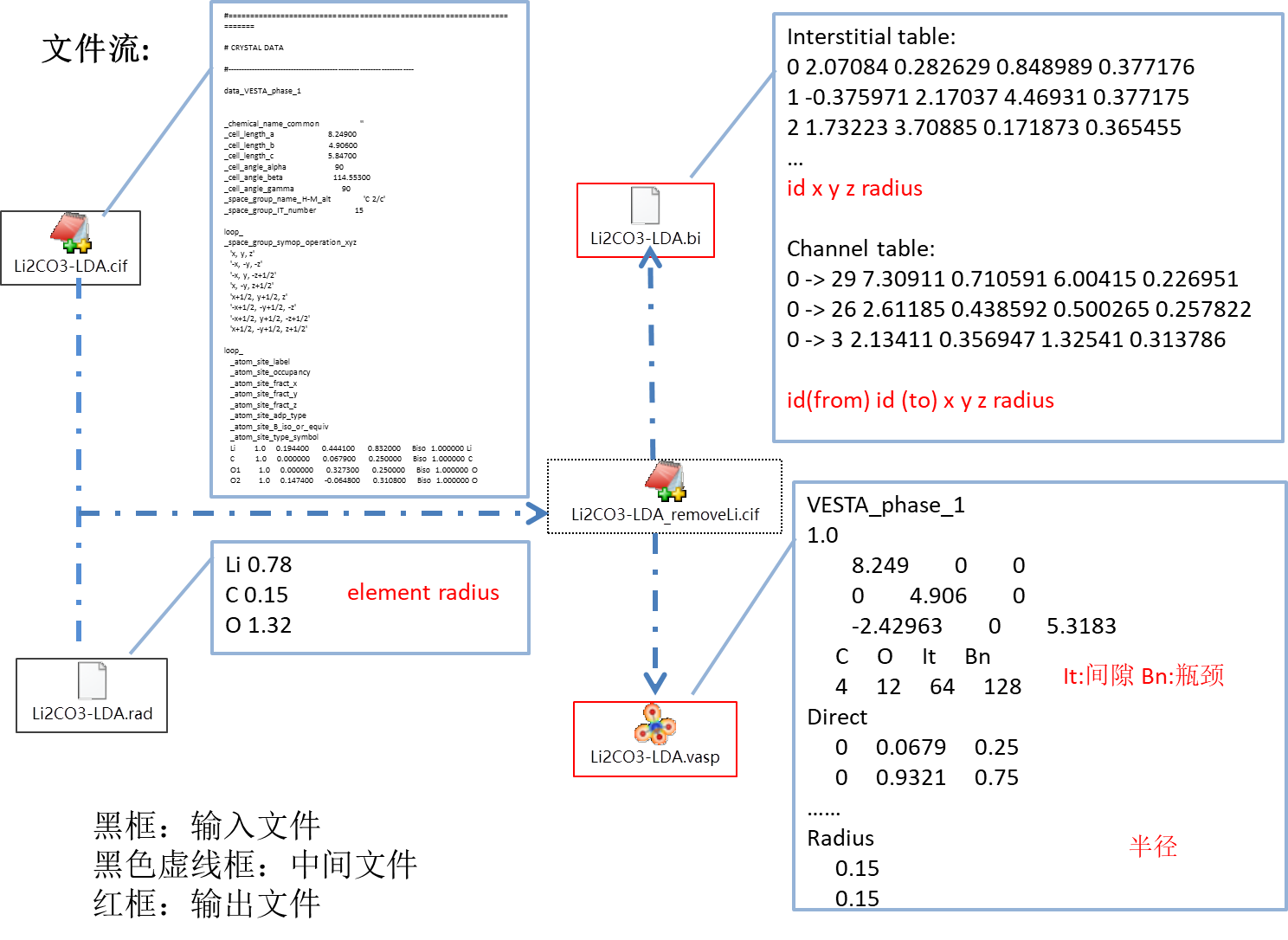
在cmd输入 python test.py运行（注意：若机器同时安装python 2.x版本和python 3.x版本，需在此处选择python 3.x版本对应的命令），结果如下：



**运行结果：**当前目录生成Li2CO3-LDA\_removeLi.cif， Li2CO3-LDA.vasp， Li2CO3-LDA.bi文件。



**文件流：**

****

### zeo. EffectiveRadCom

zeo. EffectiveRadCom(filename):

**功能：**

**参数：**

**输入：**

**输出：**

**示例：**计算半径为0.5的球形探针能够在当前目录下Li2CO3-LDA.cif结构中自由穿行，使用带半径的计算并指定自定义的半径文件**。**

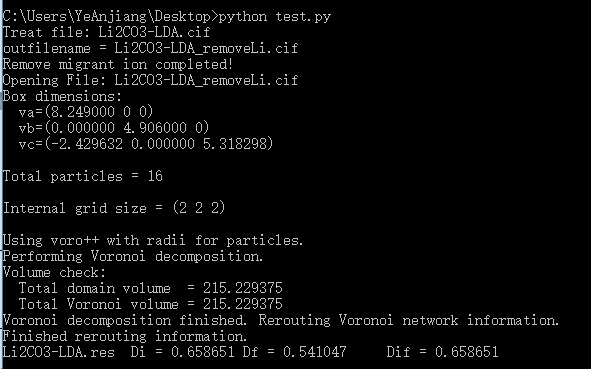
代码如下：

# test.py

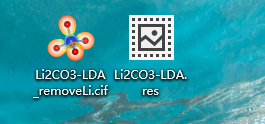
import zeo

zeo.Connection("Li2CO3-LDA.cif",0.5, "Li", True, "Li2CO3-LDA.rad")

在cmd输入 python test.py运行（注意：若机器同时安装python 2.x版本和python 3.x版本，需在此处选择python 3.x版本对应的命令），结果如下：



**运行结果：**当前目录生成Li2CO3-LDA\_removeLi.cif， Li2CO3-LDA.res文件。



### zeo. ConnValCom

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

**功能：**

**参数：**

probe\_size：输入的探针半径。

migrant：迁移离子，用于指定迁移离子，在指定情况下，程序会先去除所有的迁移离子生成一个新的cif文件，再读入这个新的cif文件进行后续计算。缺省值为None，即计算不去除迁移离子的结构中所有间隙与瓶颈。

rad\_flag：是否使用半径进行瓶颈计算。True，使用半径；False，不适用半径。

rad\_file：在rad\_flag为True时，判定是否指定自定义半径文件，如指定则使用指定的半径文件中的半径信息，否则使用程序默认半径。半径文件需命名为与输入结构文件相同的前缀名，后缀为“.rad”；文件中的内容格式为：每行指定一个元素的半径。缺省值为None。

**输入：**[.cif文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.cif)，[.rad文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.rad)。

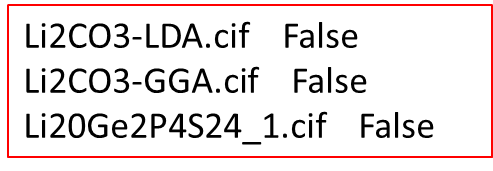
**输出：**会在指定目录下生成”results”文件夹内有一组[.res文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA.res)，），[\*\_remove[ION].cif](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA_removeLi.cif)文件（在去除迁移离子后生成的中间文件）和一个[connection\_result.txt文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/connection_result.txt)（保存探针能否通过每个结构文件的信息）。

**示例：计算半径为0.5的球形探针能够在当前目录下cifs文件夹中所有cif文件所表示的结构中自由穿行，使用带半径的计算并使用默认半径。**

import zeo

zeo.BIComputation\_batch(“./cifs/”, 0.5, “Li”, True, ”Li2CO3-LDA.rad”)

运行结果：会在当前目录生成”results”文件夹，内有connection\_result.txt文件， Li2CO3-LDA\_removeLi.cif， Li2CO3-LDA.res， 等文件。



### zeo. ConnValListCom

def ConnValListCom(filename, migrant=None, rad\_flag=True, effective\_rad=True, rad\_file=None):**功能：**从输入的.cif文件中读取结构信息，根据输入的参数，计算出该结构中瓶颈与间隙的位置以及尺寸、连通性信息（Df、Di、Dif）并保存在.vasp文件、.bi文件和.res文件中，炳并会返回Di、Df、Dif的值。

**参数：**

filename：输入的cif文件名，此文件中的结构信息是后续瓶颈与间隙计算的基础。

migrant：迁移离子。在指定某离子为迁移离子的情况下，程序会先去除cif文件中所有的迁移离子的位置信息，并生成一个新的cif文件（中间文件）。之后程序会读入这个新的cif文件，获取其中的结构信息进行后续的瓶颈与间隙计算。缺省值为None，程序直接读取输入的cif文件中的结构信息进行后续的瓶颈与间隙计算。

rad\_flag：半径标志位，表示是否使用半径进行瓶颈和间隙计算。rad\_flag = True，表明使用半径进行计算；rad\_flag = False，表明不使用半径进行计算。

rad\_file：在rad\_flag为True时，程序会判定是否给定输入的半径文件（.rad文件）。如果指定了半径文件则使用半径文件中的半径信息参与后续瓶颈和间隙计算，否则使用程序默认半径进行后续计算。半径文件需与输入结构文件具有相同的前缀名，后缀为“.rad”；文件中的内容格式为：每行指定一个元素的半径。rad\_file的缺省值为None。

rad\_store\_in\_vasp：指示是否在输出的vasp文件中保存半径信息。如果rad\_store\_in\_vasp = True，则保存半径信息；如果rad\_store\_in\_vasp = False，则不保存半径信息。

minRad：用于筛选计算得到的瓶颈与间隙的下限。当瓶颈和间隙的尺寸小于minRad时，则不保存。但当设置minRad=0和maxRad=0同成立时，保存所有瓶颈和间隙信息。

maxRad：用于筛选计算得到的瓶颈与间隙的上限。当瓶颈和间隙的尺寸大于minRad时，则不保存。但当设置minRad=0和maxRad=0同成立时，保存所有瓶颈和间隙信息。

**输入：**[.cif文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.cif)，[.rad文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.rad)。

**输出：**\*\_orgin[.bi文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA.bi)（保存瓶颈和间隙的原始信息），\*\_orgin[.vasp文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA.vasp)（保存晶体结构信息和计算得到的瓶颈和间隙的原始信息），\*\_selected.[vasp文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA.vasp)（保存晶体结构信息和经过筛选后的瓶颈和间隙的信息），\*.res文件（保存参与计算的结构的连通性信息Di，Df，Dif），[\*\_remove[ION].cif](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA_removeLi.cif)文件（[ION]为迁移离子，去除迁移离子后生成的中间文件，无实际意义）。

**返回值：**Di、Df、Dif

**示例：**计算当前目录下Li2CO3-LDA.cif结构中，Li离子迁移过程中的瓶颈和间隙，考虑半径并指定自定义的半径文件，保留半径在0.3-0.5范围内的瓶颈和间隙信息。

代码如下：

# test.py

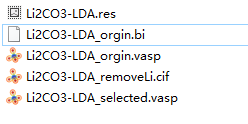
import zeo

import zeo

zeo.Computation("./tests/cifs/Li2CO3-LDA.cif",migrant="Li",rad\_flag=True, rad\_file=None, rad\_store\_in\_vasp=True, minRad=0.3, maxRad=0.5)

在cmd输入 python test.py（注意：若机器同时安装python 2.x版本和python 3.x版本，需在此处选择python 3.x版本对应的命令）运行

**运行结果：**当前目录生成Li2CO3-LDA\_removeLi.cif， Li2CO3-LDA\_orgin.vasp，Li2CO3-LDA\_selected.vasp，Li2CO3-LDA\_orgin.bi，Li2CO3-LDA.res文件。



### zeo. ConnStatusCom

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

**功能：**从输入的.cif文件中读取结构信息，根据输入的参数，计算出在当前探针尺寸下的通道和可访问表面区域信息，并将结果并保存在对应文件中。

**参数：**

filename：输入的cif文件名，此文件中的结构信息是后续瓶颈与间隙计算的基础。

probe\_rad：探针半径。

num\_sample：每个原子撒点数。

**输入：**[.cif文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.cif)，[.rad文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.rad)。

**输出：** .zcha文件，.zsa文件。

**示例：**

### zeo. ConnStatus

**功能：**

**参数：**

**输入：**

**输出：**

**示例：**计算半径为0.5的球形探针能够在当前目录下Li2CO3-LDA.cif结构中自由穿行，使用带半径的计算并指定自定义的半径文件**。**

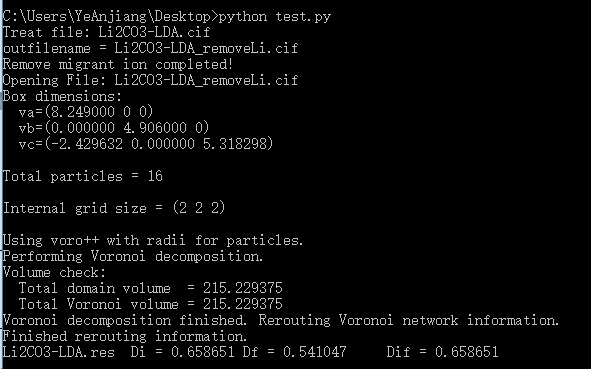
代码如下：

# test.py

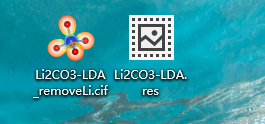
import zeo

zeo.Connection("Li2CO3-LDA.cif",0.5, "Li", True, "Li2CO3-LDA.rad")

在cmd输入 python test.py运行（注意：若机器同时安装python 2.x版本和python 3.x版本，需在此处选择python 3.x版本对应的命令），结果如下：



**运行结果：**当前目录生成Li2CO3-LDA\_removeLi.cif， Li2CO3-LDA.res文件。



### zeo. ChannelCom

**功能：**

**参数：**

**输入：**

**输出：**

**示例：**计算半径为0.5的球形探针能够在当前目录下Li2CO3-LDA.cif结构中自由穿行，使用带半径的计算并指定自定义的半径文件**。**

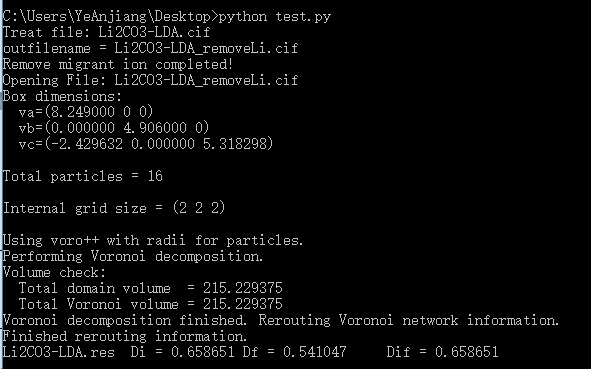
代码如下：

# test.py

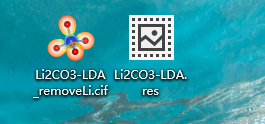
import zeo

zeo.Connection("Li2CO3-LDA.cif",0.5, "Li", True, "Li2CO3-LDA.rad")

在cmd输入 python test.py运行（注意：若机器同时安装python 2.x版本和python 3.x版本，需在此处选择python 3.x版本对应的命令），结果如下：



**运行结果：**当前目录生成Li2CO3-LDA\_removeLi.cif， Li2CO3-LDA.res文件。



### zeo. ASACom

**功能：**

**参数：**

**输入：**

**输出：**

**示例：**计算半径为0.5的球形探针能够在当前目录下Li2CO3-LDA.cif结构中自由穿行，使用带半径的计算并指定自定义的半径文件**。**

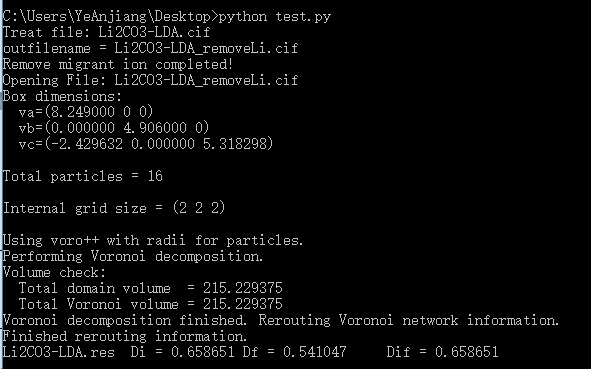
代码如下：

# test.py

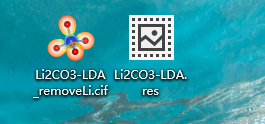
import zeo

zeo.Connection("Li2CO3-LDA.cif",0.5, "Li", True, "Li2CO3-LDA.rad")

在cmd输入 python test.py运行（注意：若机器同时安装python 2.x版本和python 3.x版本，需在此处选择python 3.x版本对应的命令），结果如下：



**运行结果：**当前目录生成Li2CO3-LDA\_removeLi.cif， Li2CO3-LDA.res文件。



### zeo. VoidNetCom

**功能：**

**参数：**

**输入：**

**输出：**

**示例：**计算半径为0.5的球形探针能够在当前目录下Li2CO3-LDA.cif结构中自由穿行，使用带半径的计算并指定自定义的半径文件**。**

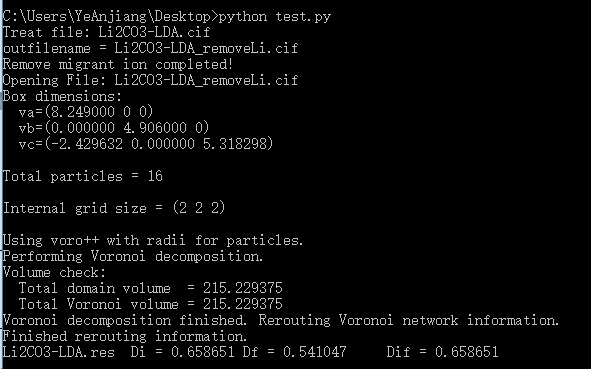
代码如下：

# test.py

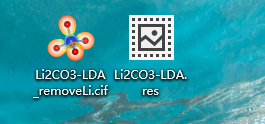
import zeo

zeo.Connection("Li2CO3-LDA.cif",0.5, "Li", True, "Li2CO3-LDA.rad")

在cmd输入 python test.py运行（注意：若机器同时安装python 2.x版本和python 3.x版本，需在此处选择python 3.x版本对应的命令），结果如下：



**运行结果：**当前目录生成Li2CO3-LDA\_removeLi.cif， Li2CO3-LDA.res文件。



### zeo. AllCom

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):

**功能：**用于计算输入的目录下所有.cif文件所表示结构中的间隙与瓶颈。

**参数：**

path：输入的目录名。

migrant：迁移离子，用于指定迁移离子，在指定情况下，程序会先去除所有的迁移离子生成一个新的cif文件，再读入这个新的cif文件进行后续计算。缺省值为None，即计算不去除迁移离子的结构中所有间隙与瓶颈。

rad\_flag：是否使用半径进行瓶颈计算。True，使用半径；False，不适用半径。

rad\_file：在rad\_flag为True时，判定是否指定自定义半径文件，如指定则使用指定的半径文件中的半径信息，否则使用程序默认半径。半径文件需命名为与输入结构文件相同的前缀名，后缀为“.rad”；文件中的内容格式为：每行指定一个元素的半径。缺省值为None。

rad\_store\_in\_vasp：指示是否在生成的vasp文件中保存半径信息。True，保存；False，不保存。

minRad：用于筛选生成的vasp文件中间隙和瓶颈，缺省值为0.0。当瓶颈和间隙的尺寸小于minRad时，则不将其保存在vasp文件中；minRad=0时，保存所有瓶颈和间隙信息。

**输入：**[.cif文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.cif)，[.rad文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.rad)。

**输出：**会在指定目录下生成”results”文件夹，内有一组[.bi文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA.bi)（保存瓶颈和间隙信息），[.vasp文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA.vasp)（保存结构和该结构中的瓶颈和间隙信息），[\*\_remove[ION].cif](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/results/Li2CO3-LDA_removeLi.cif)文件（在去除迁移离子后生成的中间文件）。

**示例：**计算当前目录下cifs文件夹中所有cif文件所表示的结构中，Li离子迁移的瓶颈和间隙，使用带半径的计算并使用默认，并将半径大于0.5的瓶颈和间隙信息保存在.vasp文件中。

import zeo

zeo.BIComputation\_batch(“./cifs/”, “Li”, True, None, True, 0.5)

运行结果：会在当前目录生成”results”文件夹，内有Li2CO3-LDA\_removeLi.cif， Li2CO3-LDA.vasp， Li2CO3-LDA.bi等文件。

## CAVD的接口使用案例

1. 计算icsd\_16713.cif

import zeo

radii = zeo.EffectiveRadCom("./icsd\_16713.cif")

print(radii)

zeo.BIComputation(filename="./icsd\_16713.cif",migrant="Li",rad\_flag=True,effective\_rad=True,rad\_file=None,rad\_store\_in\_vasp=True,minRad=0.2,maxRad=0.8)

zeo.BIComputation(filename="./Li2CO3-LDA.cif",migrant="Li",rad\_flag=False,effective\_rad=True,rad\_file=None,rad\_store\_in\_vasp=True,minRad=0.2,maxRad=0.8)

Ri,Rf,Rif = zeo.ConnValCom("./icsd\_16713.cif","Li",True,True,None)

print(Ri,Rf,Rif)

Ri1,Rf1,Rif1 = zeo.ConnValCom("./Li2CO3-LDA.cif","Li",False,True,None)

print(Ri1,Rf1,Rif1)

conn = zeo.ConnValListCom("./icsd\_16713.cif","Li",True,True,None)

conn1 = zeo.ConnValListCom("./Li2CO3-LDA.cif","Li",False,True,None)

print(conn)

print(conn1)

oneD,twoD,threeD = zeo.ConnStatusCom("./icsd\_16713.cif",0.4,"Li",True,True,None)

oneD1,twoD1,threeD1 = zeo.ConnStatusCom("./Li2CO3-LDA.cif",0.4,"Li",False,True,None)

print(oneD,twoD,threeD)

print(oneD1,twoD1,threeD1)

oneD2,twoD2,threeD2 = zeo.ConnStatus(0.4,conn)

oneD3,twoD3,threeD3 = zeo.ConnStatus(0.4,conn1)

print(oneD2,twoD2,threeD2)

print(oneD3,twoD3,threeD3)

zeo.ChannelCom("./icsd\_16713.cif",0.2,"Li",True,True,None)

zeo.ChannelCom("./Li2CO3-LDA.cif",0.2,"Li",False,True,None)

zeo.ASACom("./icsd\_16713.cif",0.5,1000,"Li",True,True,None)

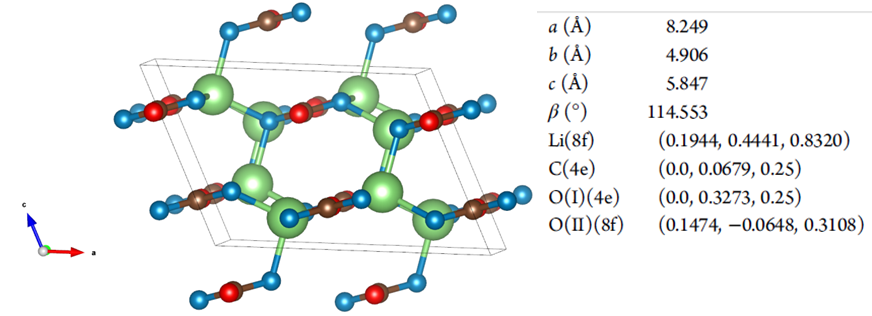
zeo.ASACom("./Li2CO3-LDA.cif",0.5,1000,"Li",False,True,None)

zeo.VoidNetCom("./icsd\_16713.cif","Li",True,True,None)

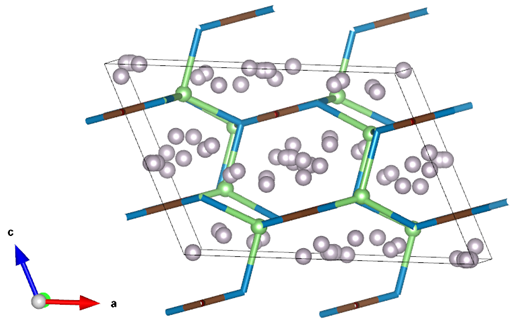
zeo.VoidNetCom("./icsd\_16713.cif","Li",False,True,None)

计算结果展示（利用vesta可视化）：

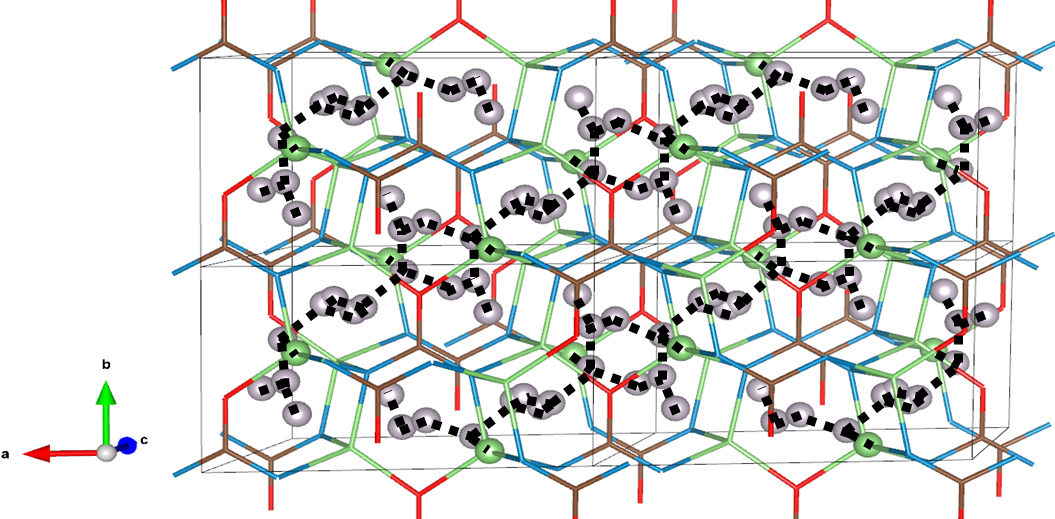
参与计算的原始结构如下：

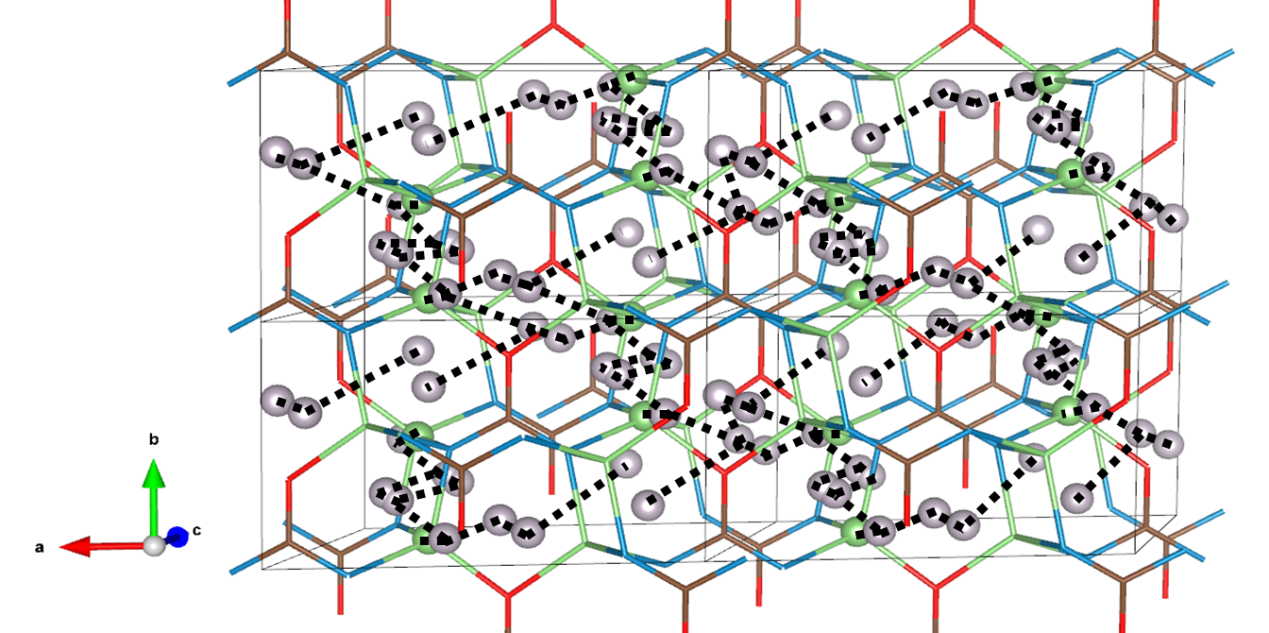


计算得到的间隙分布数据可视化后：



以r=0.584A在间隙空间中寻找通道，寻找到两条通道（2\*2\*1）：





计算得到计算得到的a，b，c方向上的Rf分别为：

* Rf\_a = 0.62105A
* Rf\_b = 0.62105A
* Rf\_c = 0.55265A

计算得到的导通性信息如下：

* 一维导通：TRUE
* 二维导通：TRUE
* 三维导通：FALSE

1. 批量计算含Li的化合物

import os

from zeo import AllCom

from zeo.netstorage import PerformVDError

from zeo.channel import FindChannelError

filenames=[]

path = "/home/yeanjiang/yaj/bi/Li\_Na\_Mg\_Al\_cifs/Li/"

if not os.path.exists(path+"results"):

os.mkdir(path+"results")

print("create results directory successful !")

else:

print(path+"results already exit!")

output\_path = path+"results/"

result\_file = open(output\_path+"com\_result\_Li.txt","w")

result\_file.write('filename\tProblem\n')

Rf\_file = open(output\_path+"Rf\_Li.txt","w")

Rf\_file.write('filename\ta\_Rf\tb\_Rf\tc\_Rf\toneD\_Conn\ttwoD\_Conn\tthreeD\_Conn\n')

for i in os.listdir(path):

if ".cif" in i:

filenames.append(i)

for filename in filenames:

filename = path+filename

try:

conn,oneD,twoD,threeD = AllCom(filename, 0.584, 1000, migrant="Li", rad\_flag=True, effective\_rad=True, rad\_file=None, rad\_store\_in\_vasp=True, minRad=0.584, maxRad=0.876)

Rf\_file.write(filename)

for i in conn:

Rf\_file.write('\t'+str(i))

Rf\_file.write('\t'+str(oneD)+'\t'+str(twoD)+'\t'+str(threeD))

Rf\_file.write("\n")

print(filename+" compute complete!")

out = filename+'\t'+'compute complete!'+'\n'

result\_file.write(out)

except AttributeError:

print(filename," Have PARTITIAL occ or MIXED occ!")

out = filename+'\t'+"Have PARTITIAL occ or MIXED occ!"+'\n'

result\_file.write(out)

continue

except IOError:

print(filename, " Can't Open inputfile or Can't Write to outputfile.")

out = filename+'\t'+"Can't Open inputfile or Can't Write to outputfile."+'\n'

result\_file.write(out)

continue

except PerformVDError:

print(filename, " Can't Perform Voronoi Decompition.")

out = filename+'\t'+"Can't Perform Voronoi Decompition."+'\n'

result\_file.write(out)

continue

except ValueError:

print(filename, " Have MIXED occ!")

out = filename+'\t'+" Have MIXED occ!"+'\n'

result\_file.write(out)

continue

except FindChannelError:

print(filename, " Can't Find Channel in Voronoi Network.")

out = filename+'\t'+"Can't Find Channel in Voronoi Network."+'\n'

result\_file.write(out)

continue

except KeyError:

print(filename, " Compute radius failed when search radius information from Shannon effective ionic radius table.")

out = filename+'\t'+"Compute radius failed when search radius information from Shannon effective ionic radius table."+'\n'

result\_file.write(out)

continue

except AssertionError:

print(filename, " Compute radius failed when pymatgen try to read information.")

out = filename+'\t'+"Compute radius failed when pymatgen try to read information."+'\n'

result\_file.write(out)

continue

except UnboundLocalError:

print(filename, " Compute radius failed when pymatgen try to compute coordnum.")

out = filename+'\t'+"Compute radius failed when pymatgen try to compute coordnum."+'\n'

result\_file.write(out)

continue

except ZeroDivisionError:

print(filename, " Integer division or modulo by zero.")

out = filename+'\t'+"Integer division or modulo by zero."+'\n'

result\_file.write(out)

continue

print("All File compute completed!")

计算结果：