基于Voronoi分解方法的晶体结构分析库

（CAVD）

V 1.0

用户手册

叶安江

20180816

目录

[1. CAVD简介 1](#_Toc528150374)

[1.1 CAVD的特点 2](#_Toc528150375)

[1.2 CAVD的主要技术 2](#_Toc528150376)

[1.3 CAVD的主要功能 3](#_Toc528150377)

[1.4 CAVD的代码架构 3](#_Toc528150378)

[2. CAVD的安装方法 4](#_Toc528150379)

[3. CAVD的计算说明 5](#_Toc528150380)

[3.1 CAVD的基本原理 5](#_Toc528150383)

[3.2 CAVD涉及的数据文件说明 9](#_Toc528150384)

[3.3 CAVD计算参数的说明 9](#_Toc528150385)

[3.4 CAVD的文件格式说明 12](#_Toc528150386)

[4. CAVD的接口说明 15](#_Toc528150387)

[4.1 cavd.EffectiveRadCom 16](#_Toc528150389)

[4.2 cavd.BIComputation 16](#_Toc528150390)

[4.3 cavd.ConnValCom 18](#_Toc528150391)

[4.4 cavd. ConnValListCom 19](#_Toc528150392)

[4.5 cavd. ConnStatusCom 20](#_Toc528150393)

[4.6 cavd. ConnStatus 21](#_Toc528150394)

[4.7 cavd. ChannelCom 22](#_Toc528150395)

[4.8 cavd. ASACom 23](#_Toc528150396)

[4.9 cavd. VoidNetCom 24](#_Toc528150397)

[4.10 cavd. AllCom 26](#_Toc528150398)

[5. CAVD的使用案例 29](#_Toc528150399)

[5.1 计算icsd\_16713.cif 29](#_Toc528150401)

[5.2 批量计算含Li的化合物 31](#_Toc528150402)

[附录1 ionic\_radii.json文件 33](#_Toc528150403)

## 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两种类型的库，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。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迁移过程中所有瓶颈和间隙的位置坐标和尺寸大小，进而可寻找迁移离子路径等信息。CAVD不仅可对单个晶体结构进行精确的分析，而且可为大型结构数据库提供高效快速的数据分析。CAVD占用的计算资源小，对每个结构的计算之间相互独立，可在一定的计算资源条件下完成更大规模的任务，并能支持多任务的提交与结果输出，这一特性恰好与高通量计算的理念不谋而合。基于CAVD的上述特性，利用CAVD作为基本计算库，辅以数据库相关知识、材料科学领域知识以及基本的晶体结构数据（cif数据、vasp数据等）可搭建高通量材料数据库平台。基于成熟的高通量材料数据库平台，后续可将计算数据进行整理归纳，更进一步的利用聚类分析等技术实现材料预测等功能。

### CAVD的特点

* 操作简单，参数设定方便。
* 计算高效，底层计算使用C++实现，计算单一结构所需耗时在秒级。
* 兼容性强，提供Python接口，方便与成熟的计算库之间相互调用。
* 内部的数据结构与用于存储的文件结构合理，方便后续的分析与功能扩展。
* 提供多种呈现数据的方式。

### CAVD的主要技术

1. 利用网络映射（network mapping）的方法表征晶体结构内部的空隙空间。晶体中各离子间的相互结合，可以看作是球体的堆积。按照晶体中质点的结合遵循势能最低的原则，从球体堆积的几何角度来看，球体堆积的密度越大，系统的势能越低，晶体越稳定。此即球体最紧密堆积原理。在离子晶体结构中，阴离子作紧密堆积，阳离子则填充在其空隙中。若要分析晶体结构中离子的迁移属性，则需要对该晶体结构中的空隙空间进行表征，最常使用的方法便是网络映射方法。
2. 通过Voronoi分解方法构建Voronoi网络映射空隙空间，并进行几何属性以及空隙拓扑的分析。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）为两个相邻离散点的垂直平分线（或面）。由于Voronoi分解法的种种特性，因此Voronoi分解法应用到了多个领域，并且都取得了相当多的成果。如：1911年，美国气象学家Alfred H. Thiessen利用Voronoi图划分陆地区域的方法用于描述天气预测，建立了Voronoi图的同义词hiessen polygons；在凝聚态物理学中，以原子为中心形成Voronoi Cell又被定义为魏格纳-塞茨原胞（Wigner-Seitz unit cells）；1934年，俄国/前苏联数学家Boris Nikolaevich Delaunay定义了Delauny三角，这与Voronoi图互为对偶图，同时利用Voronoi图以及Delauny三角为晶体结构分析提供了更强大的方法。
3. 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++进行基本的Voronoi图计算，而Voro++是一款计算Voronoi图的开源软件库，它在为一个分散的粒子系统构建Voronoi图时，不是整体的计算出整个系统的Voronoi图，而是采取基于Cell的策略，单独计算每一个粒子形成的Voronoi cell。此外，Voro++可以考虑非正交周期性边界条件，这使为具有不同晶格的晶体结构进行分析称为了可能。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计算速度。
4. 利用Cython语言将C++源代码与Python拓展模块相结合，提供了可扩展性高、调用方便的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++函数，将其封装成Python语言可调用的拓展接口模块，实现了C++与Python的结合。这种混合编程的方式，使得CAVD在运行速度与可扩展性两种不同的考量方式下找到了一个完美平衡，便于非计算机专业的客户群体使用。
5. CAVD主要是在Zeo++代码的基础上做了以下开发：

* 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在不同配位环境下的有效半径。基于准确的有效离子半径，后续进行的微观结构几何分析的准确性才有了保障。
* 用python代码实现了计算骨架离子堆积方式的程序。
* 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指定迁移离子的位置信息，从而使程序能够更方便的应用在固体电解质的分析。
* 为Zeo++中的某些功能模块编写了python接口。
* 在Zeo++代码基础上进行了适配修改，使得CAVD库实现多平台支持，可部署在Linux与Windows平台。

### CAVD的主要功能

* 晶体结构中空隙空间的表征。
* 晶体结构中间隙与瓶颈的分布与尺寸计算。
* 晶体结构中迁移离子通道计算。
* 晶体结构中迁移离子导通性计算。
* 晶体结构中迁移离子可访问区域表面积与体积计算。

### CAVD的优势分析

分析离子输运特性的方法有低精度的几何方法、键价方法，高精度的DFT-NEB、分子动力学MD方法。对单一结构，几何方法的执行时间为秒级，键价方法为数分钟，高精度方法均需要数十小时。现有的成熟数据库规模大，使用高精度方法进行逐一分析对计算资源与计算时间的要求高。执行速度快、计算代价小、流程自动的几何方法，更适合发展针对大规模数据集的高通量分析工具。

基于几何方法的离子输运分析需要根据晶体结构中的原子空间获取与其对偶的空隙空间[8]。原子空间的表征与分析已有多种成熟的方案，如晶体对称性分析工具库Spglib、原子环境模拟工具ASE、Pymatgen中的部分模块等。分析空隙空间的方案有：软件PLATON中的SOLV采用了网格法；ToposPro中使用了standard Voronoi decomposition（又称Voronoi-Dirichlet partition）方法、natural tiling方法；Zeo++，采用了radical Voronoi decomposition的方法。natural tiling方法依据简单无向连通网络的拓扑性质获取晶体空间中空隙与通道的信息，由广义的多面体（包括凸多面体和非凸多面体）组成。网格法的计算精度依赖于网格的分辨率，但分辨率提升带来的收益有限。standard Voronoi decomposition与radical Voronoi decomposition方法均为Voronoi分解方法，但前者所需的输入信息少，适用范围广，但在多分散系统中存在较大误差。radical Voronoi decomposition考虑了原子半径的，提升了多分散系统中的计算精度但需要额外输入半径表信息。

标准的Voronoi算法是一种空间分割算法，利用标准的Voronoi算法可将含N个生产元的空间完美剖分成N个Voronoi cell。Voronoi cell由生成元（中心）、顶点、边和面表征，且满足Voronoi cell中的任意点距离所包含的生成元的距离小于任何其他生成元。这一特性使得Voronoi顶点处为局部空间中最大开口处（largest opening），连接Voronoi顶点的Voronoi边为连接相邻局部空间中最大开口处的最宽路径。视晶体空间中的骨架离子为生成元，利用标准的Voronoi分解可获取由Voronoi cell（又称 Voronoi polyhedron or Wigner-Seitz cell）组成的Voronoi 图。该Voronoi图中的Voronoi顶点对应空隙空间中的局部最大间隙的中心，Voronoi边反映局部最大间隙之间的连接关系，这些Voronoi顶点与Voronoi边在局部环境中距邻居原子最远。由于相连的几何路径是晶体结构中存在离子通道的先决条件，分析由Voronoi顶点与Voronoi边组成的Voronoi网络可获取可能的离子输运通道。

晶体结构既包含原子半径相等的原子的单分散系统，也包含原子半径不等原子的多分散系统。将标准的Voronoi分解用于多分散系统，常将其中的原子视为质点再使用公式，但这种做法会带来较大误差。引入半径，使用Voronoi S分解，可准确的获取不等径晶体结构中最大开口，但Voronoi S cell的曲面边界会给后续的计算带来较大难度。为平衡计算精度与计算误差之间的平衡，CAVD引入radical Voronoi分解算法保证了计算结果近似于Voronoi S分解，计算精度高于标准Voronoi分解。

基于Voronoi分解方法进行离子输运分析的流程可总结为选择合适的Voronoi Decomposition算法、利用Voronoi图中的信息建立间隙网络模型、确定合适的几何通道判据以及定义几何/拓扑描述符。

现有的基于Voronoi方法分析离子输运通道的工作中，Blatov等人采用标准的Voronoi分解算法同时用于多分散系统以及单分散系统，Maciej开发了基于radical Voronoi算法分析多孔材料的zeo++，但底层依赖的半径表不适合分析电池材料。CAVD采用了更适合于电池材料的radical Voronoi算法，并结合配位数与香农有效离子半径表为原子空间装配更合适的半径，从而提升离子输运分析的精度。

Voronoi 网络仅包含空隙空间中最大开口处的位置与最大开口之间的连接信息，尚余利用Voronoi图中的其他信息建立间隙网络模型。CAVD用于分析迁移离子的输运通道，定义的间隙网络中包含迁移离子晶格位会为分析迁移离子之间的输运通道带来便利。Blatov和Maciej的工作只考虑在Voronoi顶点处复原迁移离子晶格位，但在具体的分析中发现，迁移离子大多复原在Voronoi顶点，但仍有部分结构中迁移离子晶格位复原在Voronoi面心中以及Voronoi路径上距周围原子最狭窄处（瓶颈）。因此，CAVD在建立间隙模型时，会在每条Voronoi边额外定义瓶颈（包含瓶颈位置与瓶颈尺寸，瓶颈尺寸为瓶颈位置距形成瓶颈的原子的距离）；在Voronoi顶点定义间隙（间隙尺寸为Voronoi顶点到离邻居原子的最短距离）；将Voronoi边定义为通道段；额外将Voronoi面心定义为间隙，并将Voronoi面心与该面上的Voronoi顶点相连加入到间隙网络中。该间隙网络是以间隙尺寸、瓶颈尺寸为权值的无相连通图，使用给定的通道判据可计算满足条件的离子输运通道。

基于硬球模型，通道判据常取某一固定的（为迁移离子的半径乘以比例），通过比较间隙尺寸、瓶颈尺寸与固定值的大小关系判断迁移离子是否可占据某间隙或在相邻的间隙间扩散。衡量一般结构弹性，为迁移离子半径的标准值。可通过统计配位环境中距离信息获取。CAVD中使用的采用剔除了LI（2769种）、Na（4761种）、Mg（1369种）、Al（2579种）配位环境下，迁移离子到骨架原子表面最短距平均值确定。、、Å、。由于不同结构弹性、配位环境等存在差异，选取相同的σ带来误差较大。CAVD使用晶胞中，迁移离子晶格位到骨架离子表面的距离与其迁移离子晶格的半径的比值定义σ：

其中，为迁移离子在晶格位与最近的骨架离子中心的距离，骨架离子的半径，为迁移离子在晶格位的半径。此外，考虑某些结构在室温难以存在通道，略微升温即可开启通道，设置=0.15作为温度容差，得到的计算公式为：

利用该公式，可自动计算任意结构中的通道判据，所有满足相连的且满足的间隙与通道段组成几何通道。

基于引入Voronoi面心的间隙网络模型以及定义的通道判据计算公式，CAVD实现了对含Li、Na、Mg、Al的6572条数据中，在99.01%的情况下完全复原了晶格中的迁移离子晶格位，并实现分析单一的结构的时间在秒级。

### CAVD的代码架构

CAVD为源代码根目录，该目录下包含6个子目录（如图所示）。

pyCavd目录中包含Cython、Python源代码以及数据文件，其中Cython源代码（.pxd文件和.pyx文件）为C++代码的封装，Python源代码（.py文件）为扩展的新功能。

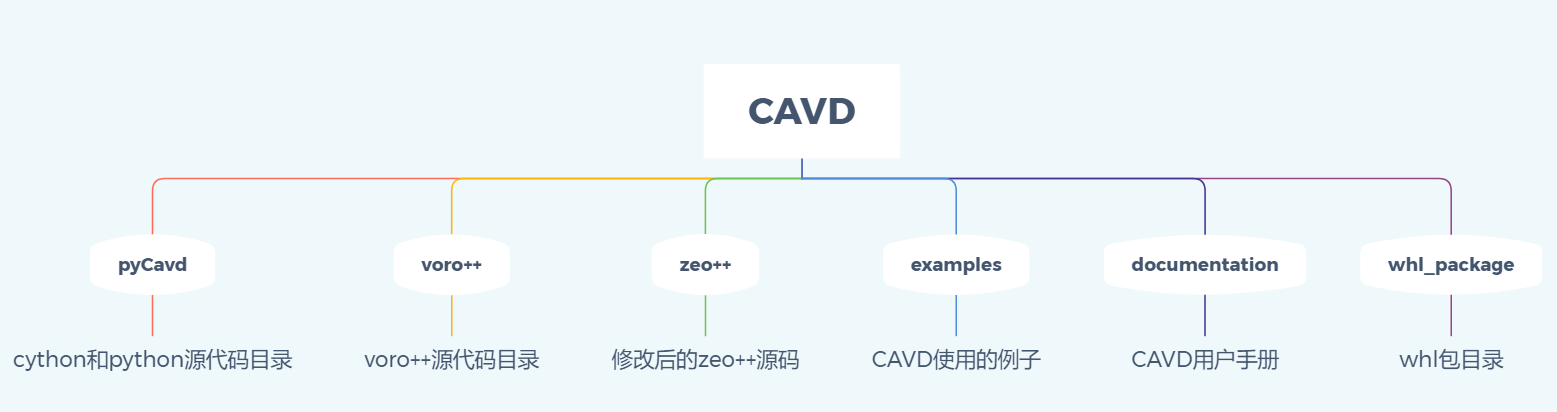
voro++目录中主要为C++代码，是voro++源代码。

zeo++目录中主要为C++代码，是在zeo++源代码基础上，结合微观结构几何分析做二次开发的代码文件。

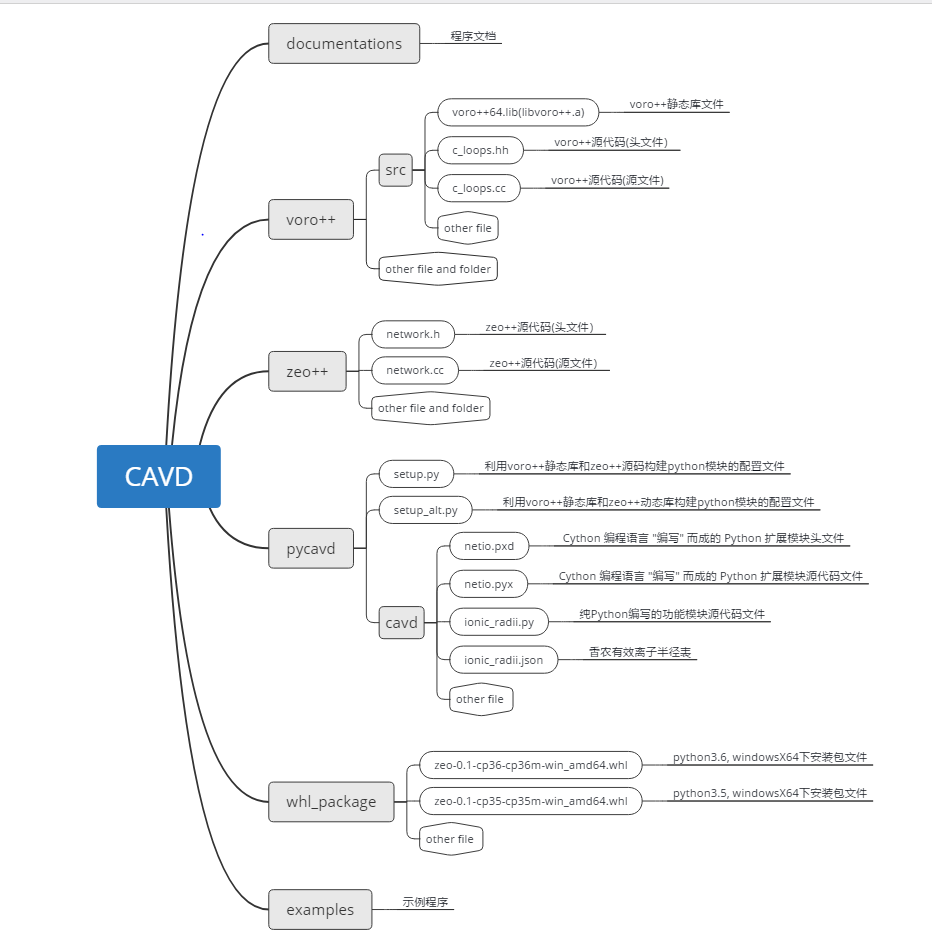
examples目录中是使用CAVD进行计算的示例。

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在不同操作系统下的安装包文件（.whl文件）。



具体的代码架构如下：



## 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”目录中包含不同环境下CAVD的.whl安装包。用户可简单使用.whl包进行安装，但当“whl\_packages”目录中安装包不满足当前环境的需求时，还可以直接从源码进行编译安装。下表为cavd在不同环境下的安装包。

|  |  |  |
| --- | --- | --- |
| 安装包名称 | 操作系统 | Python版本 |
| cavd-0.3-cp35-cp35m-win\_amd64.whl | windows X64 | Python 3.5 |
| cavd-0.3-cp36-cp36m-win\_amd64.whl | windows X64 | Python 3.6 |
| cavd-0.3-cp35-cp35m-linux\_x86\_64.whl | linux X64 | Python 3.5 |

现对安装方法做一个简单的介绍（在windows操作系统下使用whl包进行安装示范，在linux操作系统下从源码进行安装示范）：

1. windows系统下从.whl包进行安装（需提前安装依赖环境）：

|  |
| --- |
| 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 cavd-0.3-cp35-cp35m-win\_amd64.whl**  当出现Successfully installed zeo-0.3 表示安装结束。     1. 验证是否安装成功。在cmd下输入   **python**  **import cavd**  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）：

|  |
| --- |
| 1. 下载CAVD安装包。 2. 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 cavd**  若不报错，说明安装成功。 |

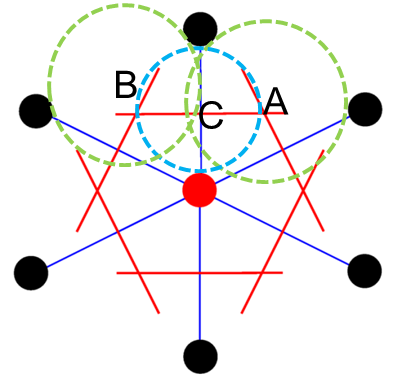
## CAVD的计算说明



### CAVD的基本原理

晶体中各离子间的相互结合，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常见方法是网络映射，即将空隙空间映射成一个网络。

完成这种映射的方法有很多种，如Natural Tilings、Voronoi Decomposition、Delaunay tessellation等方法，其中最有效的方法为Voronoi 分解方法。所谓Voronoi分解，即是将晶体结构中的空间以每个位点上的原子为中心划分成多个多面体（Voronoi cell），每个多面体由顶点（Voronoi node）和边（Voronoi edge）组成。这些多面体相互邻居且将空间完美剖分，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，且在这个多面体内任意选取一点，该点距离中心离子（原子）的距离都小于其他任何离子（或原子）的距离。每个多面体的顶点处为当前子空间中空隙较大处，多面体的边为迁移离子可能的迁移路径，迁移离子的瓶颈位于多面体的边上（如图所示）。

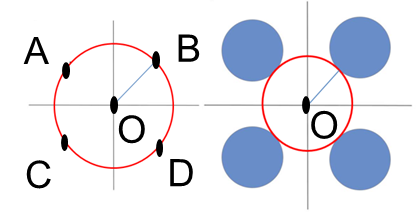


1. Voronoi Cell顶点（A、B）处构造的绿色圆半径最大。该处为可能的间隙位。
2. 在Voronoi Cell边中点处构造的蓝色圆为中心在Voronoi Cell上可构造的圆中半径最小的。该处为可能的瓶颈位。
3. 圆心沿着Voronoi Cell边可构造的圆总是较大的，故边为可能的迁移离子通道。

现就CAVD中所选取的Voronoi多面体公式与定义的详细说明如下：

1. 标准Voronoi cell定义
2. 给定的离散点集合为：;
3. 以 给出的定义对空间进行划分，所有以为生成元形成的多边形的集合：构成了离散点集合P的Voronoi图，其中的表示以离散点为中心的Voronoi Cell。

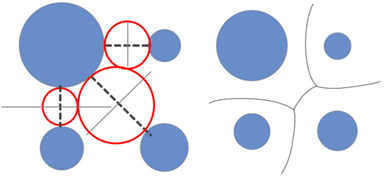
由定义可知，这种定义仅适用于质点或等径球体堆积问题，无法适用于半径不等的球体堆积问题。由这种定义进行的Voronoi 分解如图所示：



图中，黑色线表示Voronoi Cell的边，O点为Voronoi Cell的顶点，红色圆的半径为当前间隙的尺寸。以此定义求出的Voronoi cell为凸多面体且边为直线（对于三维空间中的球堆积问题则为平面），方便计算且不存在误差。

1. Voronoi S cell定义
2. 给定的球（圆）中心的集合为：，对应球（圆）的半径集合为：;
3. 以 给出的定义对空间进行划分，所有以为生成元形成的多边形的集合：构成了离散点集合P的Voronoi图，其中的表示以离散点为中心的Voronoi Cell。

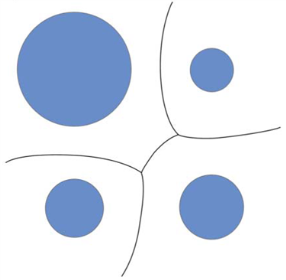
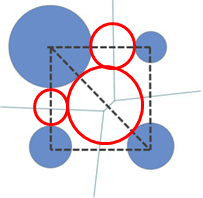
由定义可知，这种定义可用于计算半径不等的球体堆积问题。由这种定义进行的Voronoi 分解如图所示：



图中，黑色线表示Voronoi Cell的边，红色圆的半径为当前位置能放置的最大尺寸的圆。以此定义求出的Voronoi cell边为曲线（对于三维空间中的球堆积问题则为曲面），不方便计算，但这种定义下不存在误差。

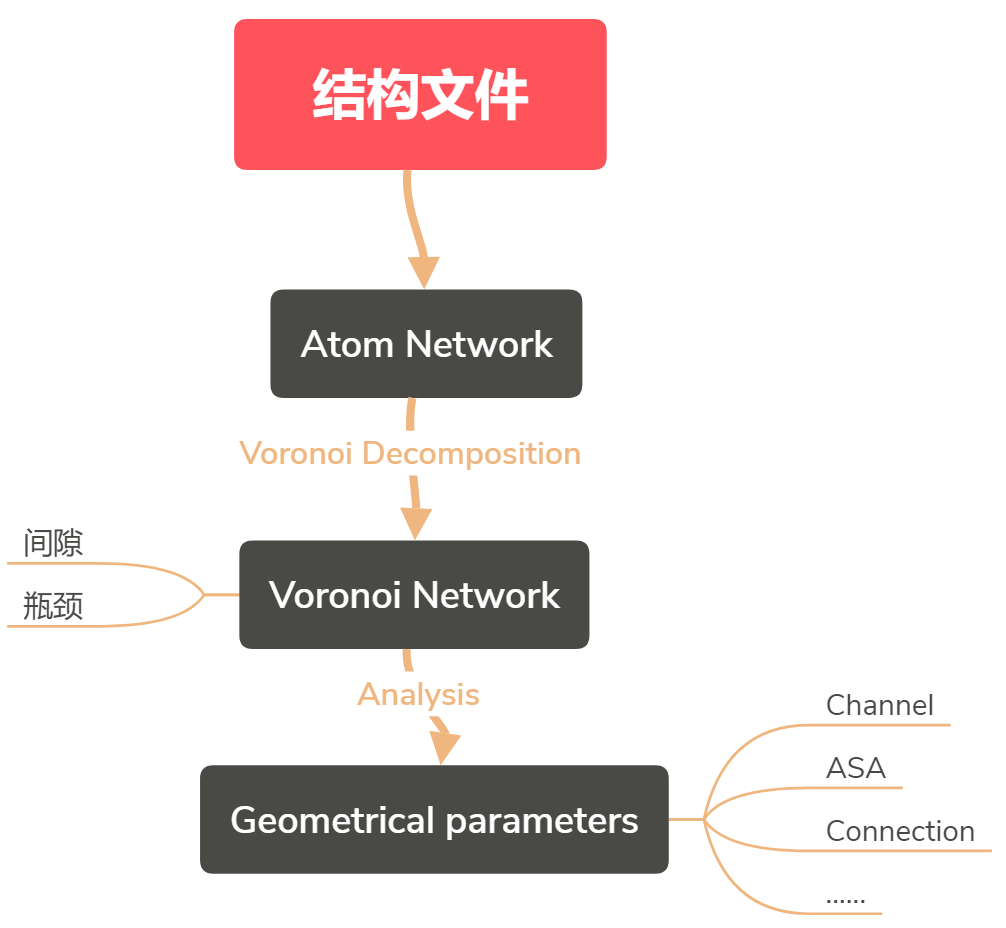
1. Radical Voronoi Cell定义
2. 给定的球（圆）中心的集合为：，对应球（圆）的半径集合为：;
3. 以 给出的定义对空间进行划分，所有以为生成元形成的多边形的集合：构成了离散点集合P的Voronoi图，其中的表示以离散点为中心的Voronoi Cell。

由定义可知，这种定义可用于计算半径不等的球体堆积问题。由这种定义进行的Voronoi 分解如图所示：



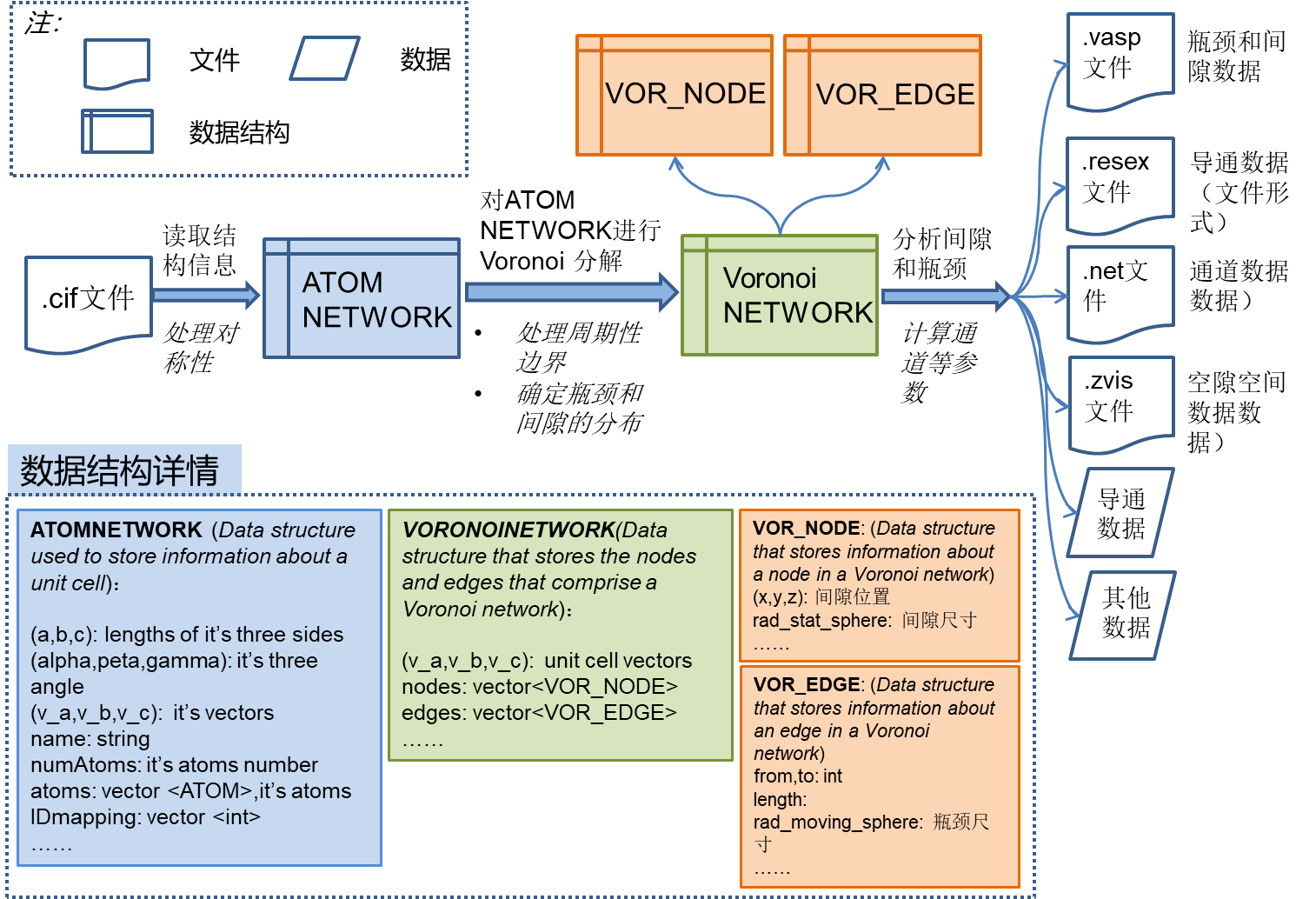
右图中，黑色线表示Voronoi Cell的边，红色圆的半径为当前位置能放置的最大尺寸的圆。以此定义求出的Voronoi cell边为直线（对于三维空间中的球堆积问题则为平面），方便计算，但这种定义下存在误差。上图右侧为间隙空间的真实分布，由Voronoi S Cell中的定义决定，上方左图为运用Radical Voronoi Cell定义求得。在此种定义下，保证了多面体边界为非凹的，且与实际分布相似（经过实验证实两者的误差很小），做到了计算效率与准确性的平衡。

CAVD则是利用上述Radical Voronoi Decomposition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（或原子）表面的距离以及形成该距离的点的位置，即为瓶颈和间隙的尺寸及位置。基于这样的Voronoi网络，以边为初始路径，顶点和边上保存的距离值为代价，利用路径搜索算法（Dijkstra最小代价算法），可得到离子的迁移路径以及结构的导通性等信息。程序大致的流程如下：



1. 读入结构文件
2. 去除迁移离子
3. 利用结构文件中的对称信息得到一个晶胞内的所有位点（Atom Network）
4. 运用Radical Voronoi Decomposition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数据
6. 将计算得到的数据以各种文件格式输出

CAVD软件的具体计算流程如下图所示：



### CAVD涉及的数据文件说明

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.），见附录一。

### CAVD计算参数的说明

1. 有效离子半径的计算方法：

首先计算给定结构文件中对应位点的配位数，然后依据元素、价态和配位数三者的信息来查找1976年香农有效离子半径表（见附录1 ionic\_radii.json文件），得到对应的半径。其中配位数的计算方法为：以目标离子为中心，10A为半径画球形区域，得到该区域内的所有的离子坐标并按照距离目标离子中心的距离排序。按照距离的远近顺序，遍历每个离子并判断它们的正负，直到识别到第一个同号离子时停止遍历，此时记录下的异号离子数目即为配位数。

1. 瓶颈尺寸定义：

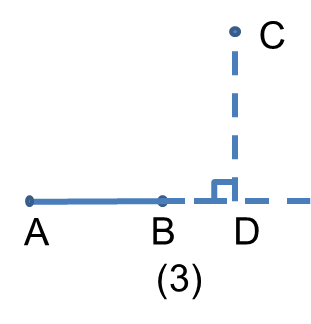
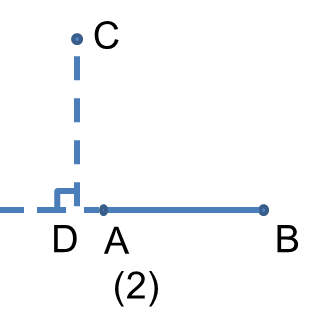
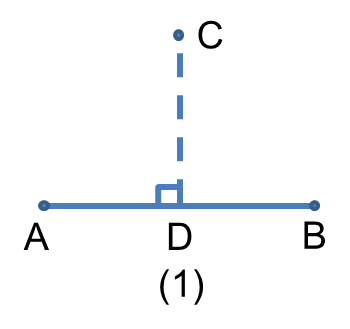
穿过瓶颈区时迁移离子到该区顶点离子表面的距离的最小值。瓶颈尺寸也即Voronoi 多面体边的与最邻近原子表面的距离，瓶颈位置则是边与最邻近原子形成最短距离所对应点的位置。CAVD对于瓶颈的计算是基于边的，也即分别计算每条边上瓶颈。转换成数学语言而言，即求解点到线段的最短距离以及线段上形成这个最短距离所对应点的位置。

对于点C和线段AB而言，C到线段AB的最短距离有(1),(2),(3)三种情况，O点为线段上形成这个最短距离所对应点的位置（如图所示）：

(1)过C作线段AB的垂线，垂足落在AB之间的点D。DC即为点C到线段最短距离，D点为线段上形成最短距离对应的点。

(2)过C作线段AB的垂线，垂足落在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点为线段上形成最短距离对应的点。

(3)过C作线段AB的垂线，垂足落在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点为线段上形成最短距离对应的点。



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。

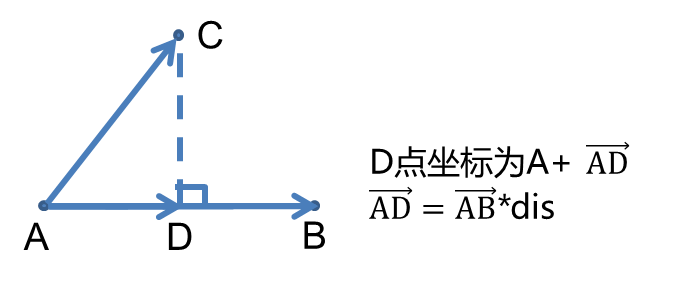
对于向量与的点积 (为向量与的夹角)，表示向量与向量在向量上投影（向量在向量上投影为）的乘积。

= ，则dis为在向量上投影与向量长度的比值，分析dis可知：

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。如图所示：



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, 最短距离即为CO长度。

1. 间隙尺寸定义

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是垂直平分面的交点，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. 通道算法

设置探针的半径为当前迁移离子的有效半径。

1. 计算给定结构一个晶胞内的周期性Voronoi 网络。
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. Ri、Rf、Rif定义

Ri为最大包含球体的半径值，定义为当前Voronoi网络所有边和结点与最近邻原子表面的距离中的最大值。

Rf为最大自由球体的半径值，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。

Rif为通道上最大包含球体的半径值，定义为组成通道的所有边和结点与最近邻原子表面的距离中的最大值。

1. 导通性定义

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的半径值（Rf\_a，Rf\_b，Rf\_c），再将给定的迁移离子有效半径R与上述三个方向上的Rf进行比较，即可判断出一维、二维、三维导通性：

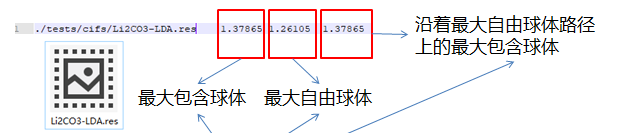
当R<Rf\_a或R<Rf\_b或R<Rf\_c，则为一维导通；

当R小于Rf\_a、Rf\_b、R<Rf\_c中的两个值，则为一维导通、二维导通；

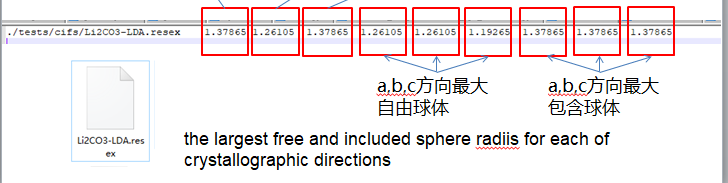
当R<Rf\_a且R<Rf\_b且R<Rf\_c，则为一维导通、二维导通、三维导通。

### CAVD的文件格式说明

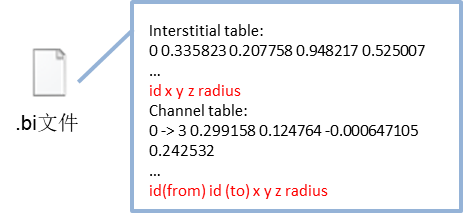
1. \*.res文件：连通性文件



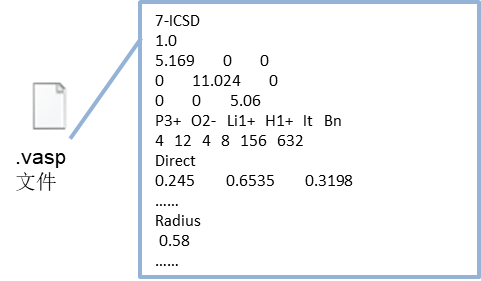
1. \*.resex文件：连通性文件



1. \*.bi文件：间隙和瓶颈的位置和尺寸信息

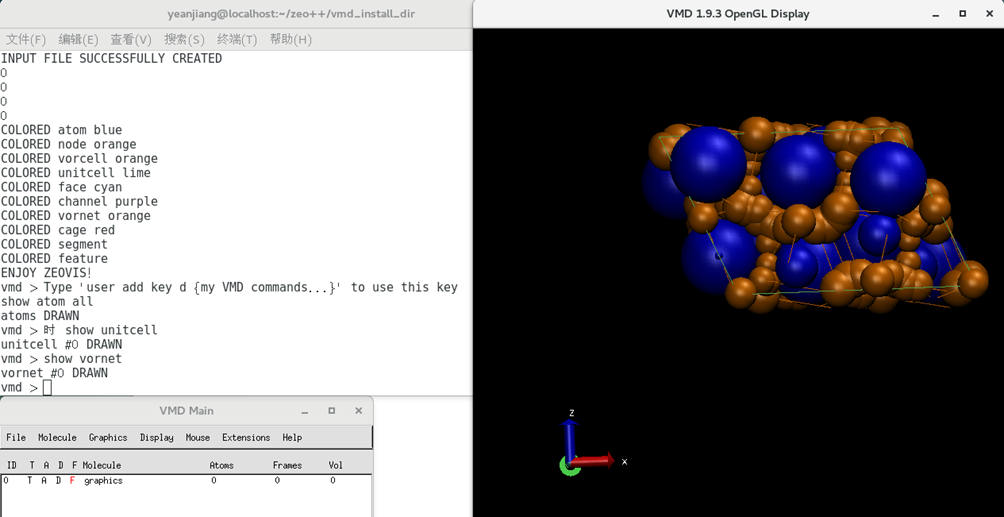


1. \*.vasp文件：瓶颈、间隙尺寸和位置信息（可视化数据；其中\*\_selected.vasp为按尺寸筛选后的信息）

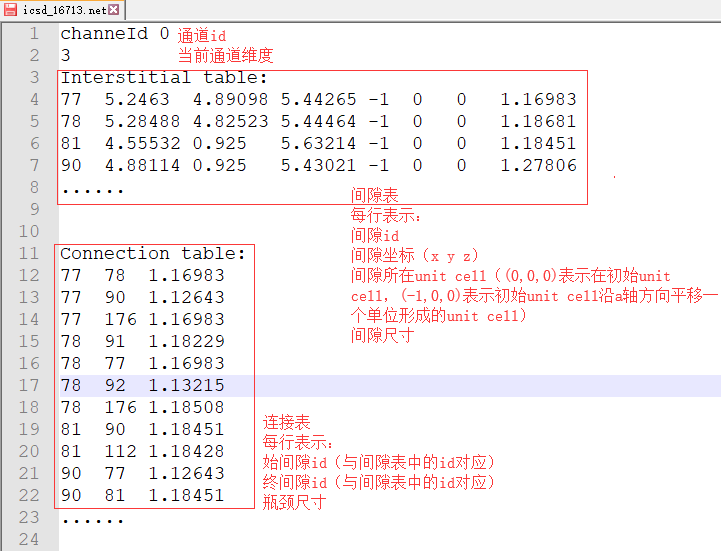


1. \*.zvis文件：Voronoi network数据文件（用于VMD的可视化数据；后续可自定义数据结构）

在VMD中显示的icsd\_16713.cif的空隙空间与结构如下所示：

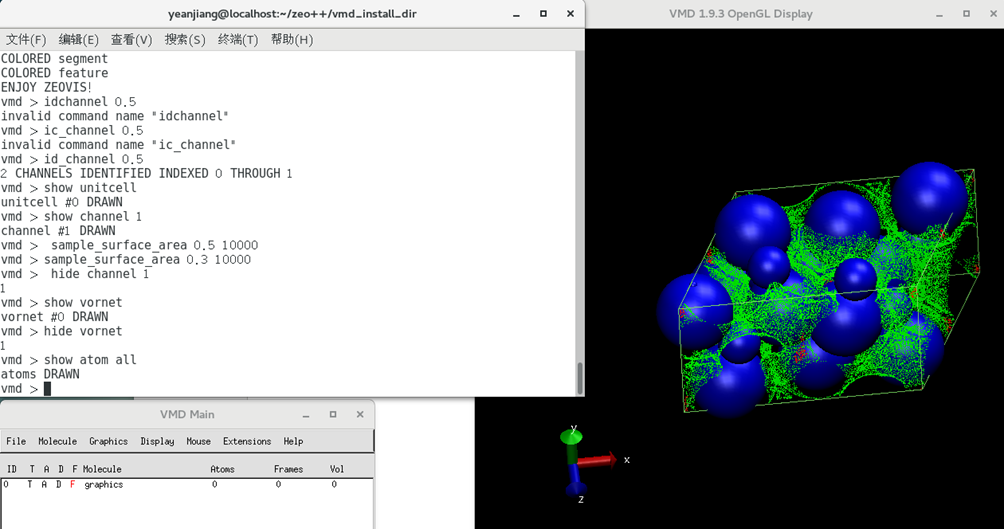
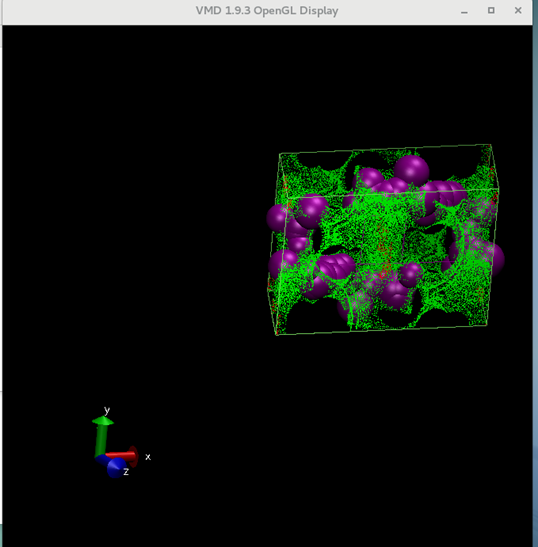


1. \*.net文件：通道数据文件。

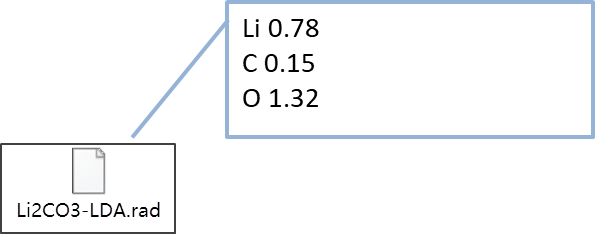


1. \*.zsa文件：可访问区域数据文件（用于VMD的可视化数据）

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：

* cavd. EffectiveRadCom()：计算输入的.cif文件所表示的结构中各种离子的半径。
* cavd.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。
* cavd.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
* cavd. 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
* cavd. 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
* cavd. ConnStatus()：计算输入的.cif文件所表示的结构中的迁移离子通道
* cavd.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。
* cavd. ASACom ()：计算输入的.cif文件所表示的结构中的可访问表面积。
* cavd. VoidNetCom()：计算输入的.cif文件所表示的结构中的空隙网络。
* cavd.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。



### cavd.EffectiveRadCom

def EffectiveRadCom(filename):

**功能：**从输入的.cif文件中读取结构信息，计算出该结构中每个离子的有效半径。

**参数：**

filename：输入的cif文件名。

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

**输出：**离子、离子有效半径形成的字典。

**示例：**计算Li2CO3-LDA.cif结构中所有离子的有效半径。

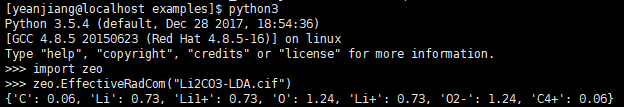
实现代码如下：

# testRad.py

import cavd

cavd.EffectiveRadCom ("Li2CO3-LDA.cif")

运行结果如下：



### cavd.BIComputation

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):

**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。

migrant：迁移离子符号。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计算。迁移离子可以缺省值，这样程序将直接读取输入的cif文件中的结构信息进行后续的计算。

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。

effective\_rad：有效离子半径标志位，表示是否使用有效离子半径进行瓶颈和间隙计算。effective \_flag = True，表明使用有效离子半径进行计算；effective \_flag = False，表明不使用有效离子半径进行计算。

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（可见前述文件格式说明）。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minRad与maxRad同时为0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上限。当瓶颈和间隙的尺寸大于maxRad时，则不保存。但当minRad与maxRad同时为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)（可缺省）。

**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）。

**示例：**计算Li2CO3-LDA.cif结构中，Li离子迁移过程中的瓶颈和间隙，考虑半径进行计算并指定自定义的半径文件，在计算得到的结果文件中保留半径0.5-0.7A之间的瓶颈和间隙。

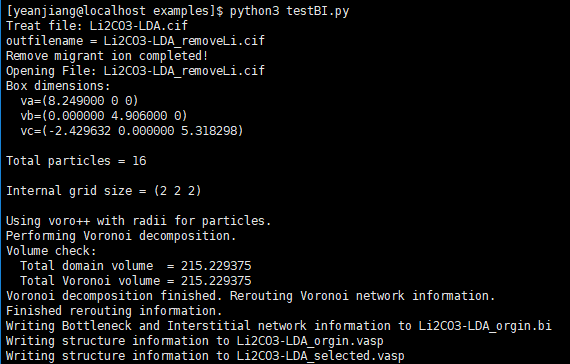
实现代码如下：

# testBI.py

import cavd

cavd.BIComputation(“Li2CO3-LDA.cif”, “Li”, True, False, ”Li2CO3-LDA.rad”, True, 0.5, 0.7)

运行结果如下：



当前目录生成Li2CO3-LDA\_orgin.bi，Li2CO3-LDA\_orgin.vasp，Li2CO3-LDA\_selected.vasp文件。

### cavd.ConnValCom

def ConnValCom(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，计算某个结构的Ri，Rf，Rif。

**参数：**

filename：输入的cif文件名。

migrant：迁移离子符号。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计算。迁移离子可以缺省值，这样程序将直接读取输入的cif文件中的结构信息进行后续的计算。

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。

effective\_rad：有效离子半径标志位，表示是否使用有效离子半径进行瓶颈和间隙计算。effective \_flag = True，表明使用有效离子半径进行计算；effective \_flag = False，表明不使用有效离子半径进行计算。

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。

**输入：**[.cif文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.cif)，[.rad文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.rad)（可缺省）。

**输出：**.res文件；返回Ri, Rf, Rif。

**示例：**计算Li2CO3-LDA.cif结构中的Ri，Rf，Rif。

实现代码如下：

# testConnVal.py

import cavd

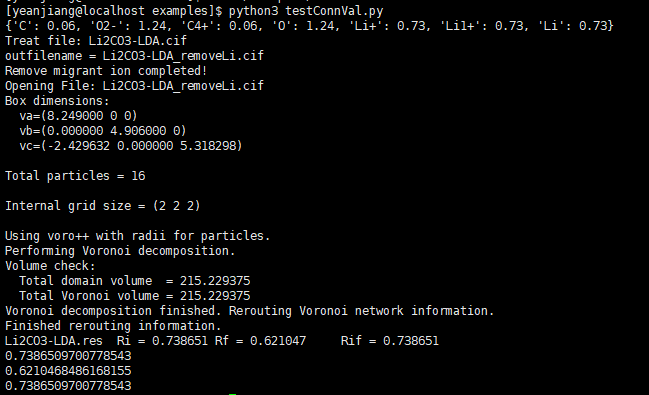
Ri,Rf,Rif = cavd.ConnValCom("Li2CO3-LDA.cif","Li",True,True,None)

print(Ri)

print(Rf)

print(Rif)

运行结果如下：



当前目录生成Li2CO3-LDA.res文件。

### cavd. 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，计算某个结构的导通性半径列表，列表中分别存放a，b，c方向上的Rf。

**参数：**

filename：输入的cif文件名。

migrant：迁移离子符号。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计算。迁移离子可以缺省值，这样程序将直接读取输入的cif文件中的结构信息进行后续的计算。

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。

effective\_rad：有效离子半径标志位，表示是否使用有效离子半径进行瓶颈和间隙计算。effective \_flag = True，表明使用有效离子半径进行计算；effective \_flag = False，表明不使用有效离子半径进行计算。

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。

**输入：**[.cif文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.cif)，[.rad文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.rad)（可缺省）。

**输出：**.ressex文件；导通半径列表（a、b、c方向上的Rf组成的列表）。

**示例：**计算当前Li2CO3-LDA.cif结构中a、b、c方向上的Rf。

实现代码如下：

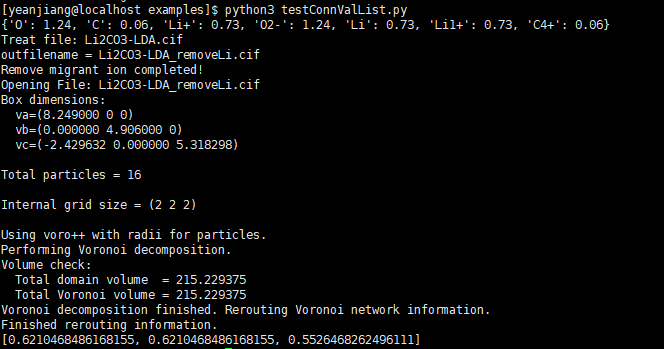
# testConnValListCom.py

import cavd

conn = cavd.ConnValListCom("Li2CO3-LDA.cif","Li",True,True,None)

print(conn)

运行结果如下：



当前目录生成Li2CO3-LDA.resex文件。

### cavd. 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，根据输入的参数和目标迁移离子的半径，判断它是否是1D，2D，3D导通。

**参数：**

filename：输入的cif文件名。

radius：给定目标离子的半径，单位为A。

migrant：迁移离子符号。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计算。迁移离子可以缺省值，这样程序将直接读取输入的cif文件中的结构信息进行后续的计算。

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。

effective\_rad：有效离子半径标志位，表示是否使用有效离子半径进行瓶颈和间隙计算。effective \_flag = True，表明使用有效离子半径进行计算；effective \_flag = False，表明不使用有效离子半径进行计算。

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。

**输入：**[.cif文件](PyBI/Zeo++/zeo/trunk/cython_wrapper/zeo/tests/cifs/Li2CO3-LDA.cif)，[.rad文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.rad)（可缺省）。

**输出：** 三个boolean值组成的列表，分别表示目标离子在a、b、c方向上的导通性。

**示例：**判断半径为0.5A的目标离子在Li2CO3-LDA.cif结构中a、b、c方向上的导通性。

实现代码如下：

# testConnStatusCom.py

import cavd

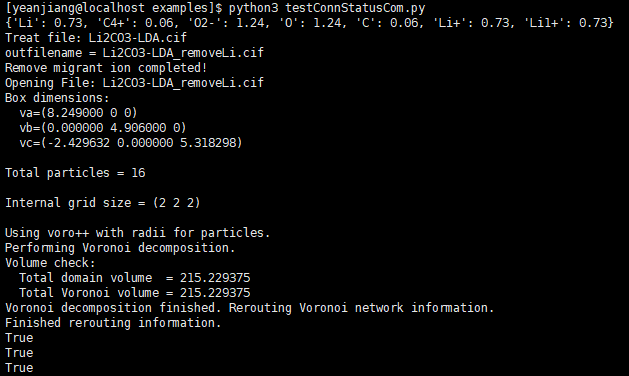
oneD,twoD,threeD = cavd.ConnStatusCom("Li2CO3-LDA.cif",0.5,"Li",True,True,None)

print(oneD)

print(twoD)

print(threeD)

运行结果如下：



当前目录生成Li2CO3-LDA.resex文件。

### cavd. ConnStatus

def ConnStatus(radius,connlist):

**功能：**从输入的.cif文件中读取结构信息，根据输入的目标迁移离子的半径值和指定结构的导通半径列表，判断它是否是1D，2D，3D导通。

**参数：**

radius：给定目标离子的半径，单位为A。

connlist：导通半径列表，由结晶学a、b、c方向上的Rf组成。

**输入：**导通半径列表

**输出：**三个boolean值组成的列表，分别表示目标离子在a、b、c方向上的导通性。

**示例：**判断半径为0.5A的目标离子在Li2CO3-LDA.cif结构中a、b、c方向上的导通性。

实现代码如下：

# testConnStatus.py

import cavd

conn = cavd.ConnValListCom("Li2CO3-LDA.cif","Li",True,True,None)

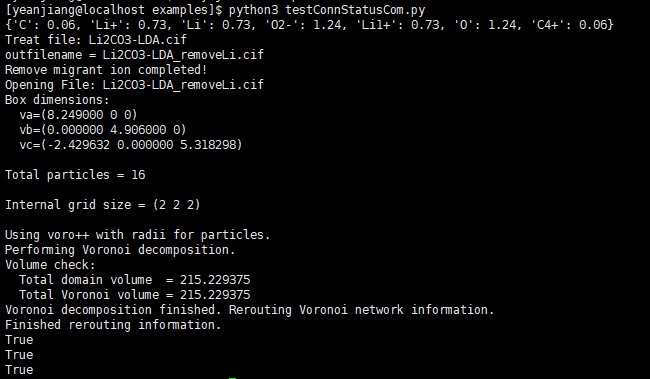
oneD,twoD,threeD = ConnStatus(0.5,conn)

print(oneD)

print(twoD)

print(threeD)

运行结果如下：



当前目录生成Li2CO3-LDA.resex文件。

### cavd. ChannelCom

def ChannelCom(filename, probe\_rad, 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，根据输入的参数和目标迁移离子的半径值计算出当前迁移离子的迁移通道。

**参数：**

filename：输入的cif文件名。

probe\_rad：给定目标离子的半径，单位为A。

migrant：迁移离子符号。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计算。迁移离子可以缺省值，这样程序将直接读取输入的cif文件中的结构信息进行后续的计算。

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。

effective\_rad：有效离子半径标志位，表示是否使用有效离子半径进行瓶颈和间隙计算。effective \_flag = True，表明使用有效离子半径进行计算；effective \_flag = False，表明不使用有效离子半径进行计算。

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。

**输入：** [.cif文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.cif)，[.rad文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.rad)（可缺省）。

**输出：**.net文件。

**示例：**计算半径为0.5A的目标离子在Li2CO3-LDA.cif结构中迁移通道。

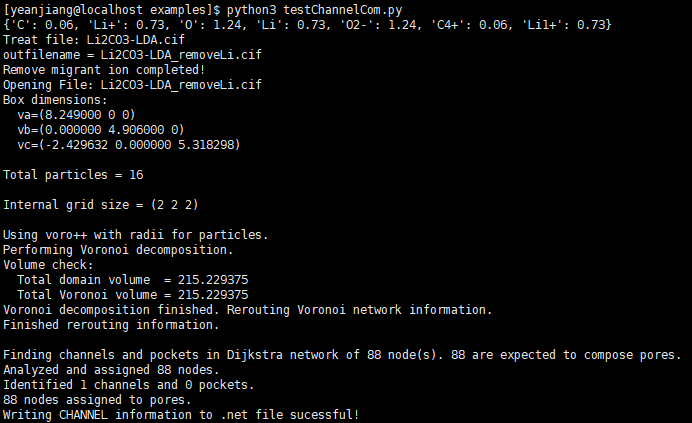
实现代码如下：

# testChannelCom.py

import cavd

cavd.ChannelCom("Li2CO3-LDA.cif",0.5,"Li",True,True,None)

运行结果如下：



当前目录生成Li2CO3-LDA.net文件。

### cavd. ASACom

def ASACom(filename, probe\_rad, num\_sample, 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，根据输入的参数和目标迁移离子的半径值计算出当前迁移离子在当前结构中的可访问区域。

**参数：**

filename：输入的cif文件名。

probe\_rad：给定目标离子的半径，单位为A。

num\_sample：分析可访问区域时的采样点数目。

migrant：迁移离子符号。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计算。迁移离子可以缺省值，这样程序将直接读取输入的cif文件中的结构信息进行后续的计算。

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。

effective\_rad：有效离子半径标志位，表示是否使用有效离子半径进行瓶颈和间隙计算。effective \_flag = True，表明使用有效离子半径进行计算；effective \_flag = False，表明不使用有效离子半径进行计算。

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。

**输入：**[.cif文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.cif)，[.rad文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.rad)（可缺省）。

**输出：**.zsa文件。

**示例：**计算半径为0.5A的目标离子在Li2CO3-LDA.cif结构中可访问的区域，取样点数设置为1000。

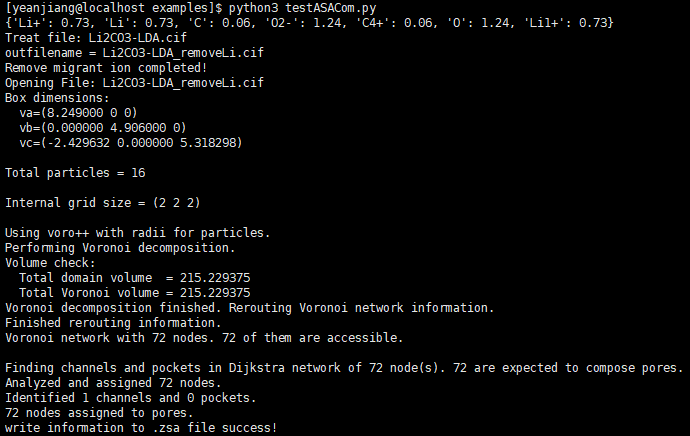
实现代码如下：

# testASACom.py

import cavd

cavd.ASACom("Li2CO3-LDA.cif",0.5,1000,"Li",True,True,None)

运行结果如下：



当前目录生成Li2CO3-LDA.zsa文件。

### cavd. VoidNetCom

def VoidNetCom(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，计算出当前结构中的空隙网络。

**参数：**

filename：输入的cif文件名。

migrant：迁移离子符号。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计算。迁移离子可以缺省值，这样程序将直接读取输入的cif文件中的结构信息进行后续的计算。

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。

effective\_rad：有效离子半径标志位，表示是否使用有效离子半径进行瓶颈和间隙计算。effective \_flag = True，表明使用有效离子半径进行计算；effective \_flag = False，表明不使用有效离子半径进行计算。

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。

**输入：** [.cif文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.cif)，[.rad文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.rad)（可缺省）。

**输出：**.zsa文件。

**示例：**计算Li2CO3-LDA.cif结构中Li间隙网络。

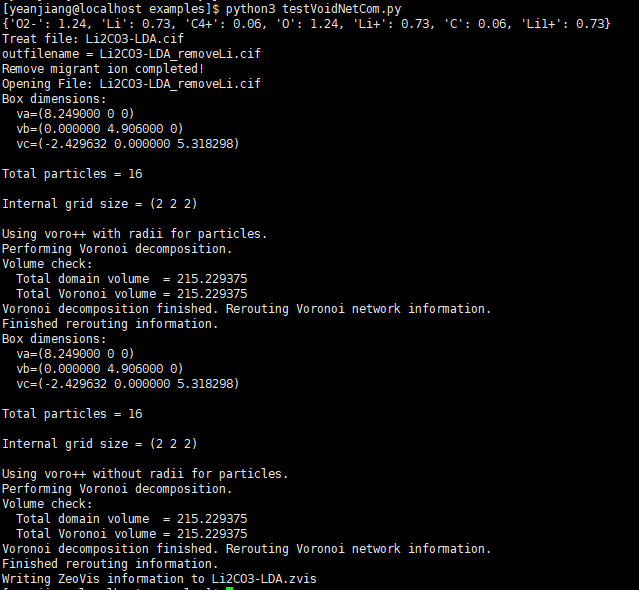
实现代码如下：

# testASACom.py

import cavd

cavd.ASACom("Li2CO3-LDA.cif",0.5,1000,"Li",True,True,None)

运行结果如下：



当前目录生成Li2CO3-LDA.zvis文件。

### cavd. 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：**从输入的.cif文件中读取结构信息，计算该结构中的瓶颈、间隙、导通值列表、导通性、通道、ASA和空隙分布网络。此函数是对cavd所有功能的集合，从而能更好的用于批量处理。

**参数：**

filename：输入的cif文件名。

probe\_rad：给定目标离子的半径，单位为A。

num\_sample：分析可访问区域时的采样点数目。

migrant：迁移离子符号。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计算。迁移离子可以缺省值，这样程序将直接读取输入的cif文件中的结构信息进行后续的计算。

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。

effective\_rad：有效离子半径标志位，表示是否使用有效离子半径进行瓶颈和间隙计算。effective \_flag = True，表明使用有效离子半径进行计算；effective \_flag = False，表明不使用有效离子半径进行计算。

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minRad与maxRad同时为0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上限。当瓶颈和间隙的尺寸大于maxRad时，则不保存。但当minRad与maxRad同时为0时，保存所有瓶颈和间隙信息。

**输入：**[.cif文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.cif)，[.rad文件](file:///F:\CAVD\documentation\PyBI\Zeo++\zeo\trunk\cython_wrapper\zeo\tests\cifs\Li2CO3-LDA.rad)（可缺省）。

**输出：** .vasp文件，.bi文件，.net文件，.resex文件，.zsa文件，.zvis文件；导通数值列表，1D、2D、3D导通性。

**示例：**计算Li2CO3-LDA.cif结构中瓶颈、间隙、导通值列表、导通性、通道、ASA和空隙分布网络。瓶颈间隙筛选范围为0.5~0.7A，目标离子半径值为0.5A，取样点数为1000。

实现代码如下：

# testAllCom.py

import cavd

conn,oneD,twoD,threeD = cavd.AllCom("Li2CO3-LDA.cif",0.5,1000,"Li",True,True,None,True,0.5,0.7)

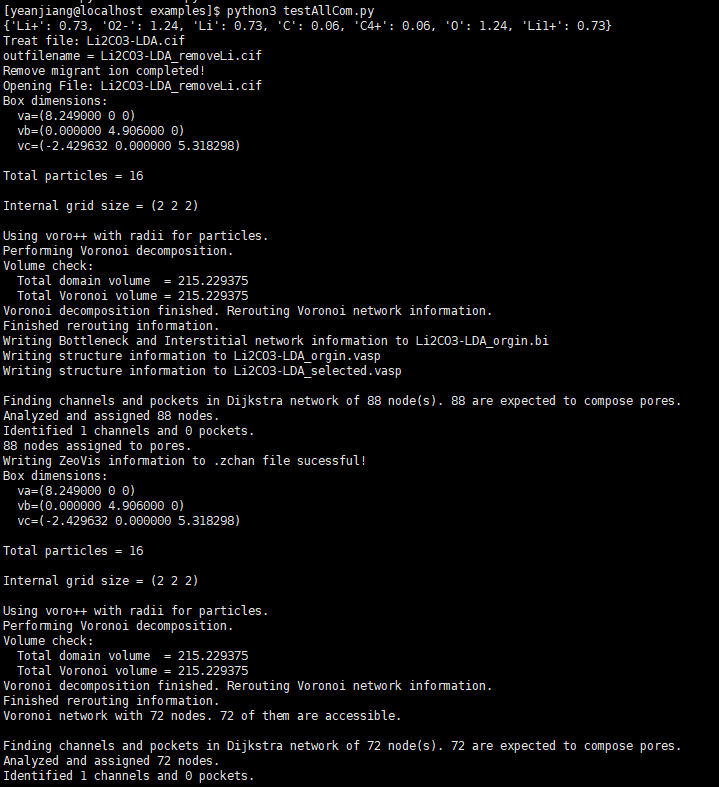
print(conn)

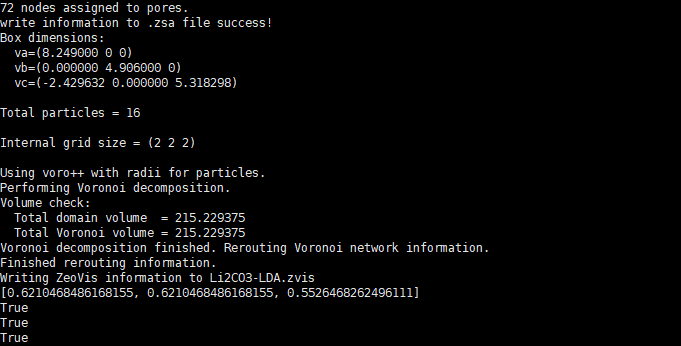
print(oneD)

print(twoD)

print(threeD)

运行结果如下：





会在当前目录生成.vasp文件，.bi文件，.net文件，.resex文件，.zsa文件，.zvis文件。

## CAVD的使用案例



### 计算icsd\_16713.cif

import cavd

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

print(radii)

cavd.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)

cavd.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 = cavd.ConnValCom("./icsd\_16713.cif","Li",True,True,None)

print(Ri,Rf,Rif)

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

print(Ri1,Rf1,Rif1)

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

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

print(conn)

print(conn1)

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

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

print(oneD,twoD,threeD)

print(oneD1,twoD1,threeD1)

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

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

print(oneD2,twoD2,threeD2)

print(oneD3,twoD3,threeD3)

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

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

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

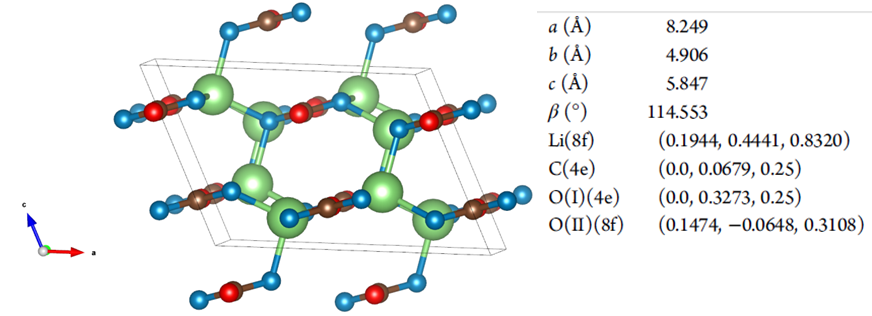
cavd.ASACom("./Li2CO3-LDA.cif",0.5,1000,"Li",False,True,None)

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

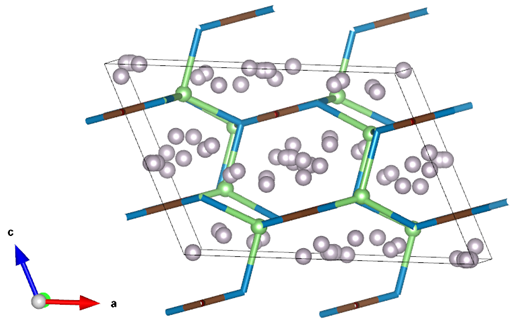
zeo.VoidNetCom("./icsd\_16713.cif","Li",False,True,None)

计算结果展示（利用vesta可视化）：

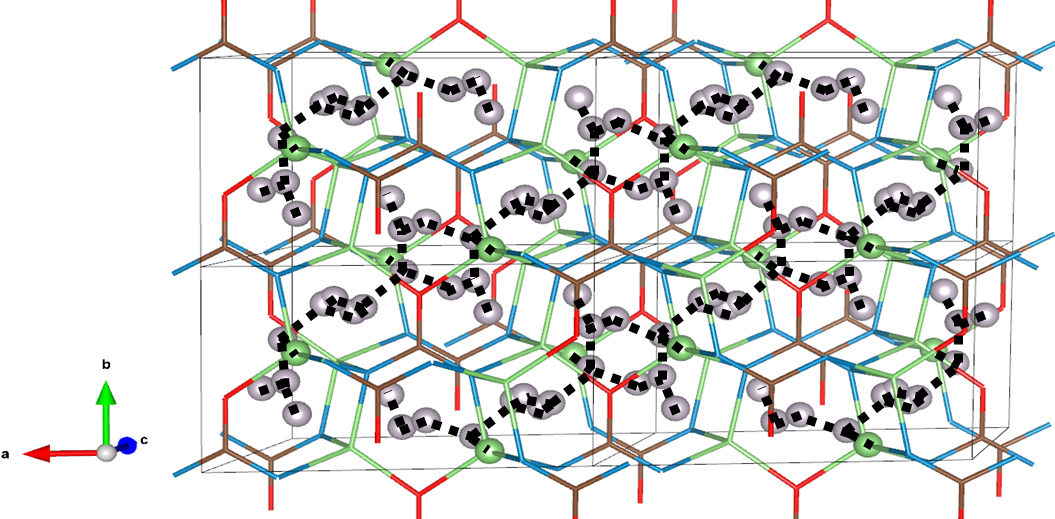
参与计算的原始结构如下：

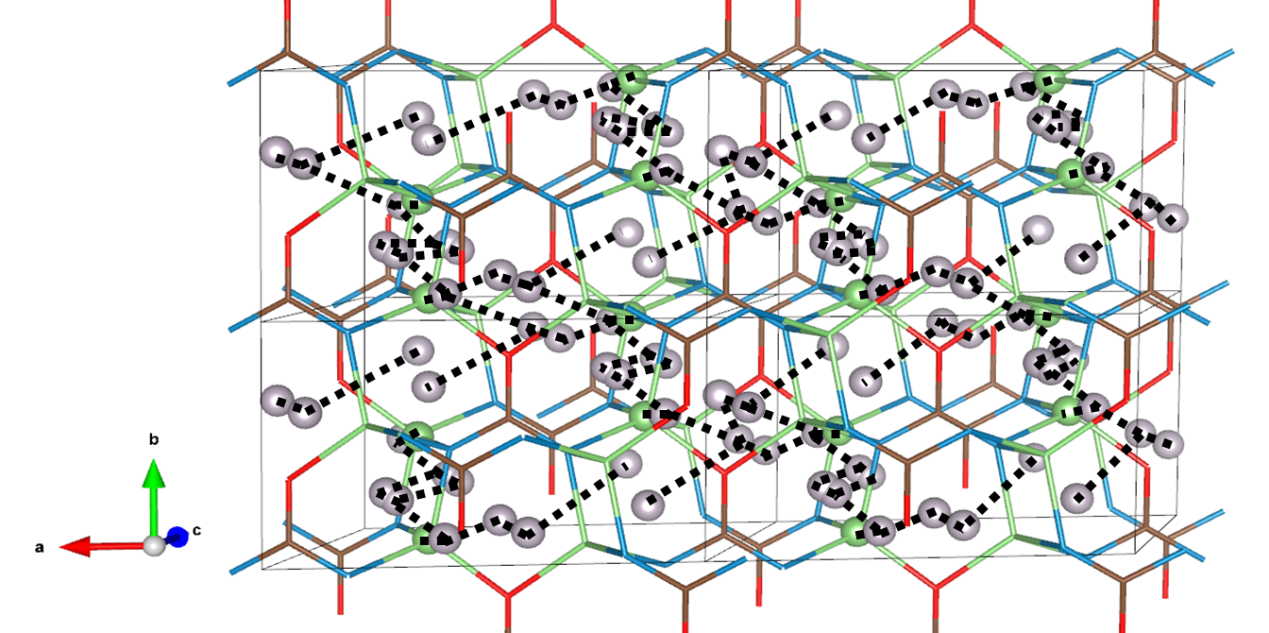


计算得到的间隙分布数据可视化后：



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

### 批量计算含Li的化合物

import os

from cavd import AllCom

from cavd.netstorage import PerformVDError

from cavd.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：

对6159种含Li化合物计算，得到瓶颈数据2769条、间隙数据2769条、通道数据2769条、导通数据2704条、空隙空间数据2704条。