

Report:DNA-transcription factor binding sites

Abstract

This report explains a local web server. In the fore-end, the users first download pdb file by inputting the pdb code. In the back-end, the DNAPRO is used to resolve pdb files, and then print part of information in the fore-end in form. For the other part of information, 3dmol is used to show it in 3d model.

1. Introduction

The implementation mainly contains two parts: (1) downloading pdb files and (2) resolving the pdb files.

2. Download of pdb files

The fore-end for downloading pdb files is shown in figure 1.

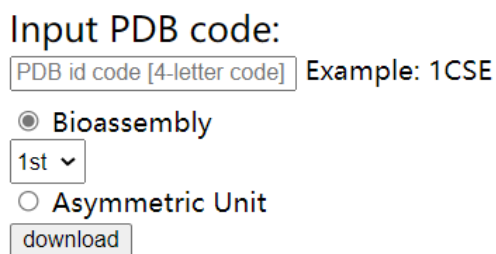


Figure 1. The fore-end for downloading pdb files.

As can be seen from the figure, there is an input box which asks input PDB code. The valid formation of PDB code is the 4-letter code and invalid input will not download the pdb file successfully.

The code snippet for the input box is shown in figure 2:

After inserting the pdb code, users can click on button: **download**, to get the corresponding pdb files. In the back-end, the inputted value in input box is first obtained by `document.getElementById("pdb_id_input1").value`, where the `pdb_id_input1` is the id of the input box. After obtaining the inputted value, the link to the pdb file should be created. The website <https://files.rcsb.org/download/pdbCode.pdb> provides

```
<div class="row">
  <div class="small-3 columns">
    <label
      for="pdb_file_input"
      class="left"
      style="font-size:1.3rem">
      Upload PDB file:
    </label>
  </div>
  <div class="small-6 columns">
    <input
      type="file"
      id="pdb_file_input"
      name="pdb_file"
      class="radius"
      onchange="fileChange(this);"/>
  </div>
  <div class="small-3 columns">
    <input
      type="text"
      id="pdb_id_input1"
      name="pdb_id1"
      style="margin-bottom:0.5rem"
      placeholder="PDB id code [4-letter code]"
      class="radius" />
  </div>
</div>
```

Figure 2. The snippet for the input box.

the pdb files, where the `pdbCode` should be replaced by the specific pdb code, namely the inputted value. And then, the pdb file can be downloaded by call `window.open()` function. The following figure 3 and figure 4 are the implementation of **download** button and the creation of download link together with the call of `window.open()` function, respectively.

```
<div class="row">
  <button
    type = "submit"
    id = "btn_submit"
    onclick = "download()">
    download
  </button>
</div>
```

Figure 3. The snippet for download button.

```

<script>
var v = document.getElementById("pdb_id_input").value
function download() {
    link1 = "https://files.rcsb.org/download/" +
    document.getElementById("pdb_id_input").value+ ".pdb"
    window.open(link1)
}
</script>

```

Figure 4. The snippet for creation of download link together with the download by the link.

3. Visualization

The users should upload the pdb files first, then resolve the file and further visualize it by **py3Dmol**.

3.1. Uploading pdb files

The code snippet for uploading pdb files is shown in figure 5.

```

<div class="row">
  <div class="small-3 columns">
    <label
      for="pdb_file_input"
      class="left"
      style="font-size:1.3rem">
      Upload PDB file:
    </label>
  </div>
  <div class="small-6 columns">
    <input
      type="file"
      id="pdb_file_input"
      name="pdb_file"
      class="radius"
      onchange="fileChange(this);"/>
    </div>
  </div>
</div>

```

Figure 5. The snippet for uploading pdb files.

3.2. Visualization of pdb files

The **py3Dmol** is used in the visualization of pdb files, which can easily display the interactions. For interactions involving a ring (pi-cation, pi-stacking), ProLIF returns the index of one of the ring atoms, but for visualisation having the centroid of the ring looks nicer. We will start by writing a function to find the centroid, given the index of one of the ring atoms. The code snippet is shown in the figure 6.

The following code allows selecting of some protein identifiers which are used to render the 3D protein structure, as shown in figure 7.

Now, let me explain the code,

- * lines[1-3]. Call to libraries, Besides **streamlit**, clearly, we have to invoke **Py3Dmol**, but we will also call a function named **showmol** from **stmol**. The function

```

from rdkit import Chem
from rdkit import Geometry

def get_ring_centroid(mol, index):
    # find ring using the atom index
    Chem.SanitizeMol(mol,
    Chem.SanitizeFlags.SANITIZE_SETAROMATICITY)
    ri = mol.GetRingInfo()
    for r in ri.AtomRings():
        if index in r:
            break
    else:
        raise ValueError("No ring containing this
        atom index was found in the given molecule")
    # get centroid
    coords = mol.xyz[list(r)]
    ctd = plf.utils.get_centroid(coords)
    return Geometry.Point3D(*ctd)

```

Figure 6. The function to find the centroid, given the index of one of the ring atoms.

```

import streamlit as st
import py3Dmol
from stmol import showmol
st.sidebar.title('Show Proteins')
prot_str='1A2C,1BML,1D5M,1D5X,1D5Z,1D6E,1DEE,
1E9F,1FC2,1FCC,1G4U,1GZS,1HE1,1HEZ,1HQR,1HXY,
1IBX,1JBU,1JWM,1JWS'
prot_list=prot_str.split(',')
bcolor = st.sidebar.color_picker('Pick A Color', '#00f900')
protein=st.sidebar.selectbox('select protein',prot_list)
style = st.sidebar.selectbox('style',['line','cross',
'stick','sphere','cartoon','clicksphere'])
spin = st.sidebar.checkbox('Spin', value = False)
xyzview = py3Dmol.view(query='pdb:'+protein)
xyzview.setStyle({'style':{'color':'spectrum'}})
xyzview.setBackgroundColor(bcolor)
if spin:
    xyzview.spin(True)
else:
    xyzview.spin(False)
xyzview.zoomTo()
showmol(xyzview,height=500,width=800)

```

Figure 7. Selecting of some protein identifiers.

showmol is a replacement for the **show** method that renders the 3D model.

- * line 4. We add a title in the sidebar.
- * lines [5,6]. We use standardized labels to identify some proteins which **py3Dmol** can query later.
- * lines [7–10]. Here we have three widgets in the sidebar, one for setting the background color, and two more select boxes to choose the protein label given in line 4, and the other to choose the style for the molecular rendering. For molecules, the more usual are **stick** and **sphere**, while for proteins the preferred is **cartoon**. Line 10 defines a checkbox for if we want to make the molecule spin.
- * lines [11–18]. In these lines, we make the **py3Dmol** object **xyzview**. In line 11 we do the query for the protein selected in line 8, and, in line 13 we set the background color selected in line 7. From lines 14 to 17, spin is activated, depending on the value in the checkbox. Line 18 makes zoom to the whole molecule.

* lines [19]. Finally, in line 19 we call **showmol** to render the resulting molecule view. This is the more important step in this code, as without this it would not be possible to view the object. I have to say that **showmol** accepts as an argument the py3Dmol object and optionally the width and the height of the window which by default has **width=500** and **height=500**.

The following code converts SMILES string into 3D molecular structures, as shown in figure 8.

```
import streamlit as st
from stmol import showmol
import py3Dmol

from rdkit import Chem
from rdkit.Chem import AllChem

st.title('RDKit + Py3DMOL 😊')

def makeblock(smi):
    mol = Chem.MolFromSmiles(smi)
    mol = Chem.AddHs(mol)
    AllChem.EmbedMolecule(mol)
    mblock = Chem.MolToMolBlock(mol)
    return mblock

def render_mol(xyz):
    xyzview = py3Dmol.view()#(width=400,height=400)
    xyzview.addModel(xyz,'mol')
    xyzview.setStyle({'stick':{}})
    xyzview.setBackgroundColor('white')
    xyzview.zoomTo()
    showmol(xyzview,height=500,width=500)

compound_smiles=st.text_input('SMILES please','CC')
blk=makeblock(compound_smiles)
render_mol(blk)
```

Figure 8. Converting SMILES string into 3D molecular structures.

Notice that the function `makeblock` returns a mol object which contains the 3D structure generated from the SMILES string. Line 14 is where we do the conversion from a 2D representation to 3D with the `MolToMolBlock` method. Also notice that the `render_mol` function is the same used before, with the only difference that in line 19, I changed the format given to the `addModel` method. This time I need it to be the mol format (others are pdb, sdf, xyz, etc.). Thus, you can change that depending on the format you are using.

Protein-ligand interactions are typically represented with the ligand in atomic details, residues as nodes, and interactions as edges. Such diagram can be eas-

ily displayed by calling ProLIF's builtin class `prolif.plotting.network.LigNetwork`. This diagram is interactive and allows moving around the residues, as well as clicking the legend to toggle the display of specific residues types or interactions. `LigNetwork` can generate two kinds of depictions:

- (1) Based on a single specific frame
- (2) By aggregating results from several frames

In the latter case, the frequency with which an interaction is seen will control the width of the corresponding edge. we can hide the least frequent interactions by using a threshold, i.e. `threshold=0.3` will hide interactions that occur in less than 30 percent of frames.