**Earlier**

Screenshots (FRED view in VIDA) of different .oedu’s, before reruns with .mtz files attached. Cavities are clearly visible.

MNPGAfbeelding met kaart

Automatisch gegenereerde beschrijving

Laura237

Afbeelding met diagram

Automatisch gegenereerde beschrijving

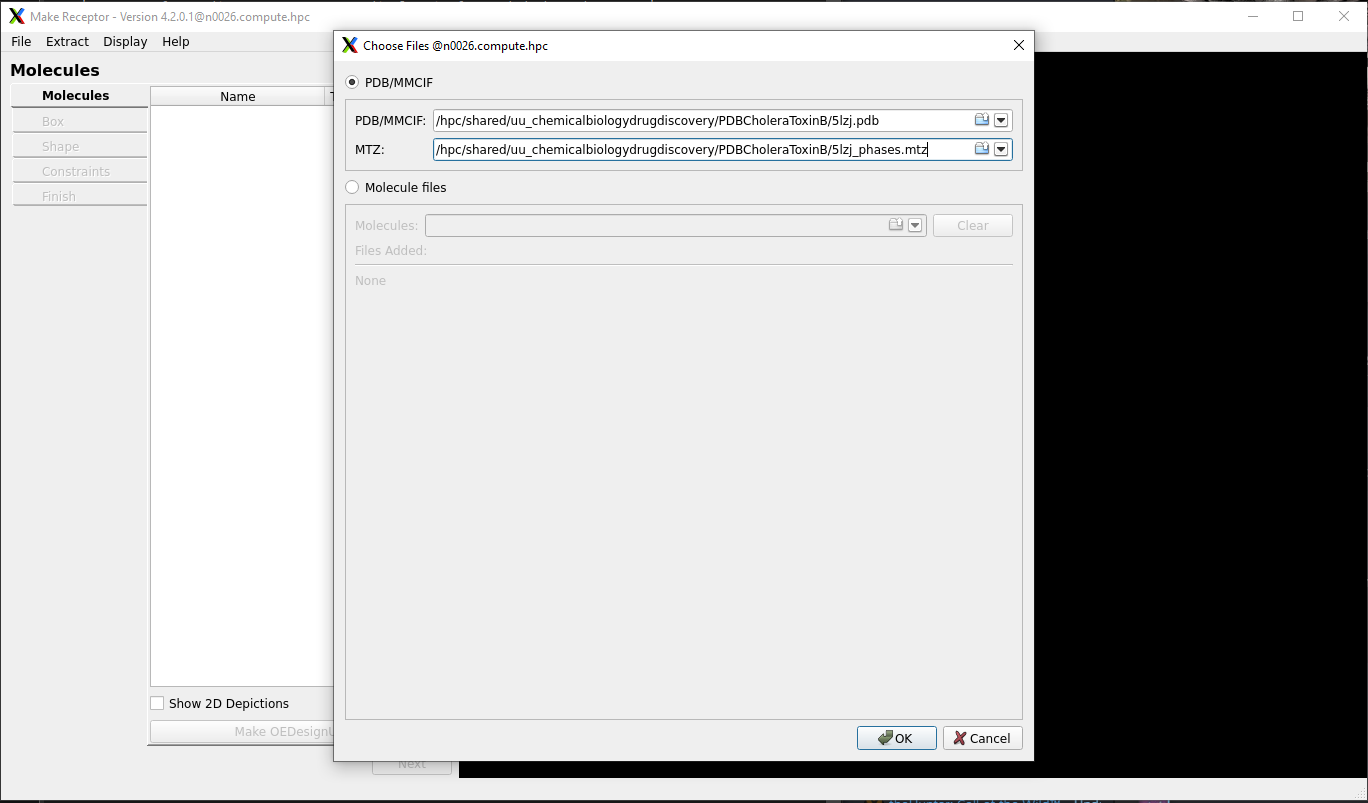
PC262

**Afbeelding met tekst, plastic

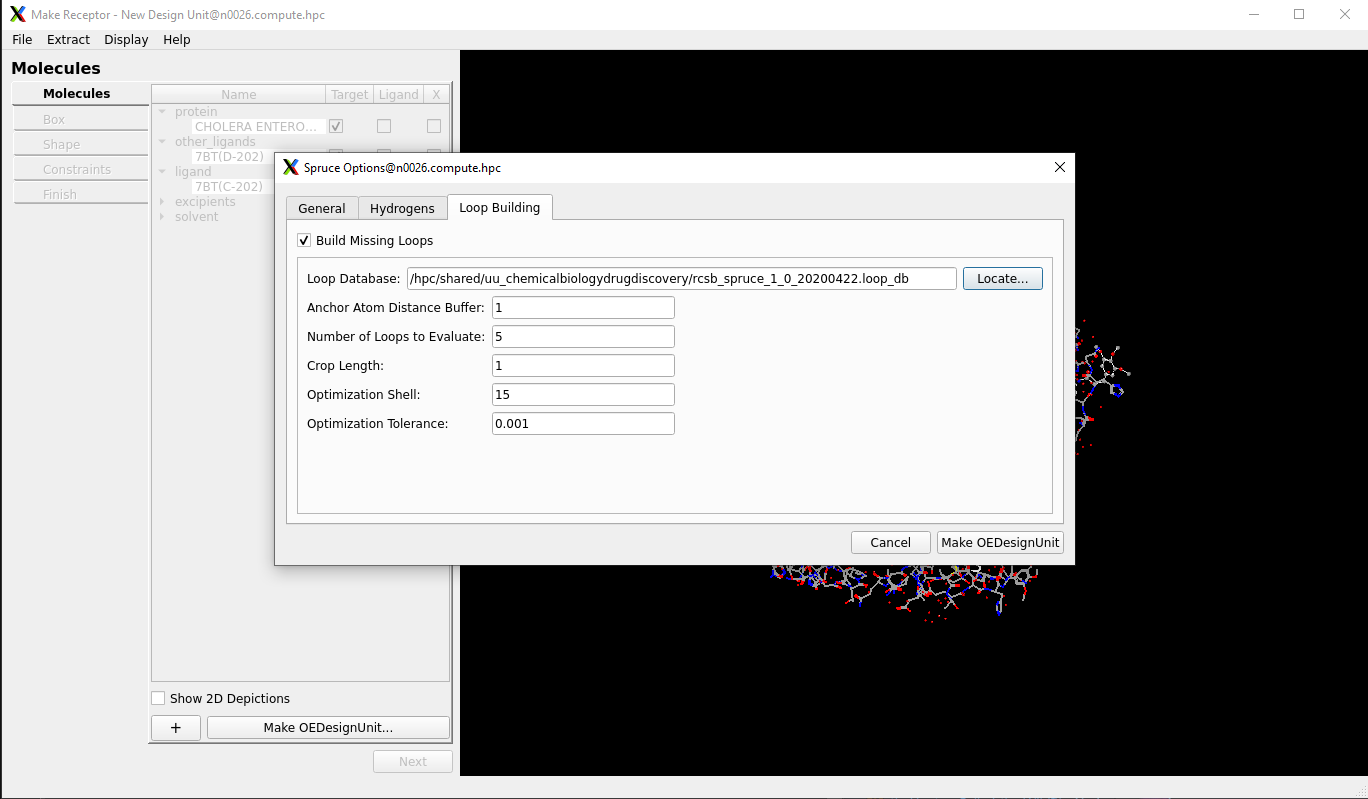
Automatisch gegenereerde beschrijving**

**Thursday 16/03/2023**

Made new .oedu for every PDB with corresponding .mtz file. 5lzj is used as example below.



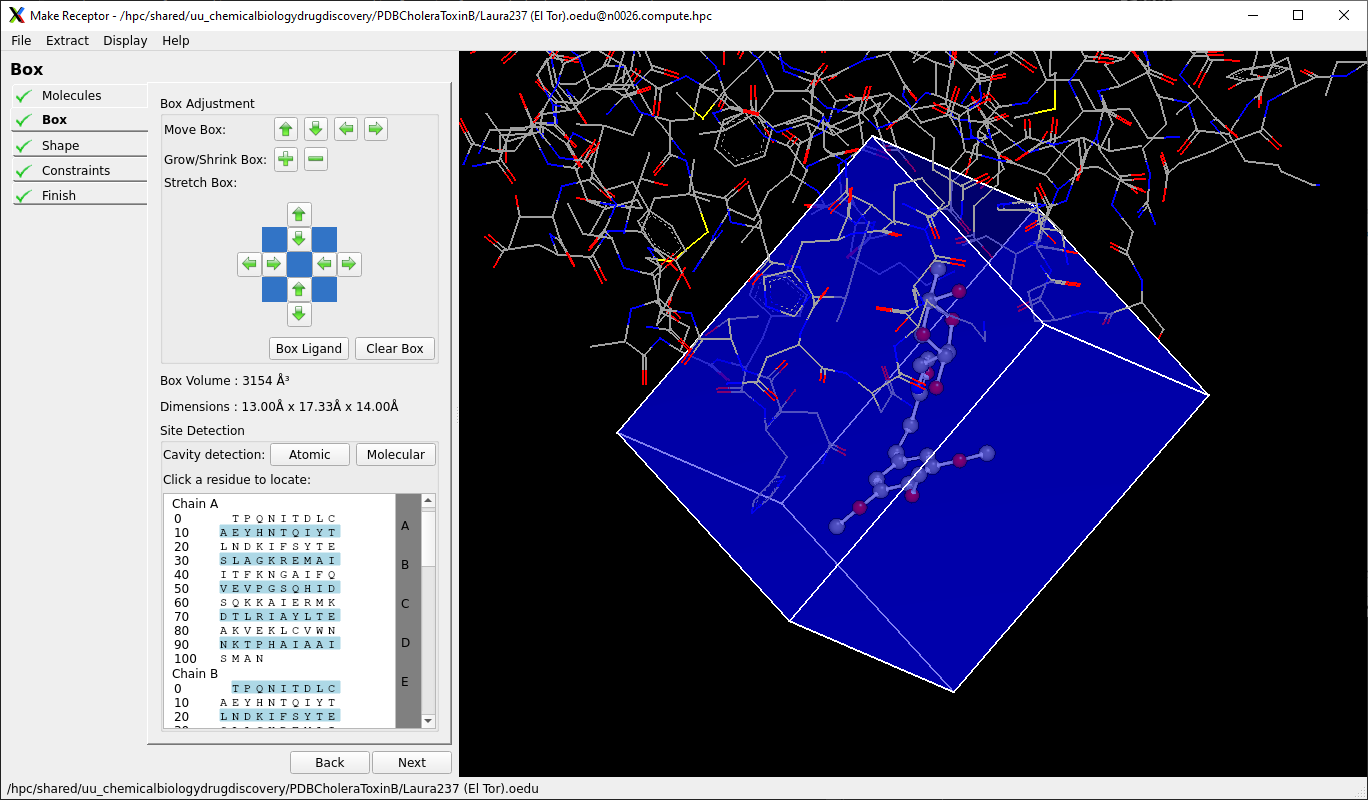
We use a loop database obtained at openeye, everything else is at default settings.



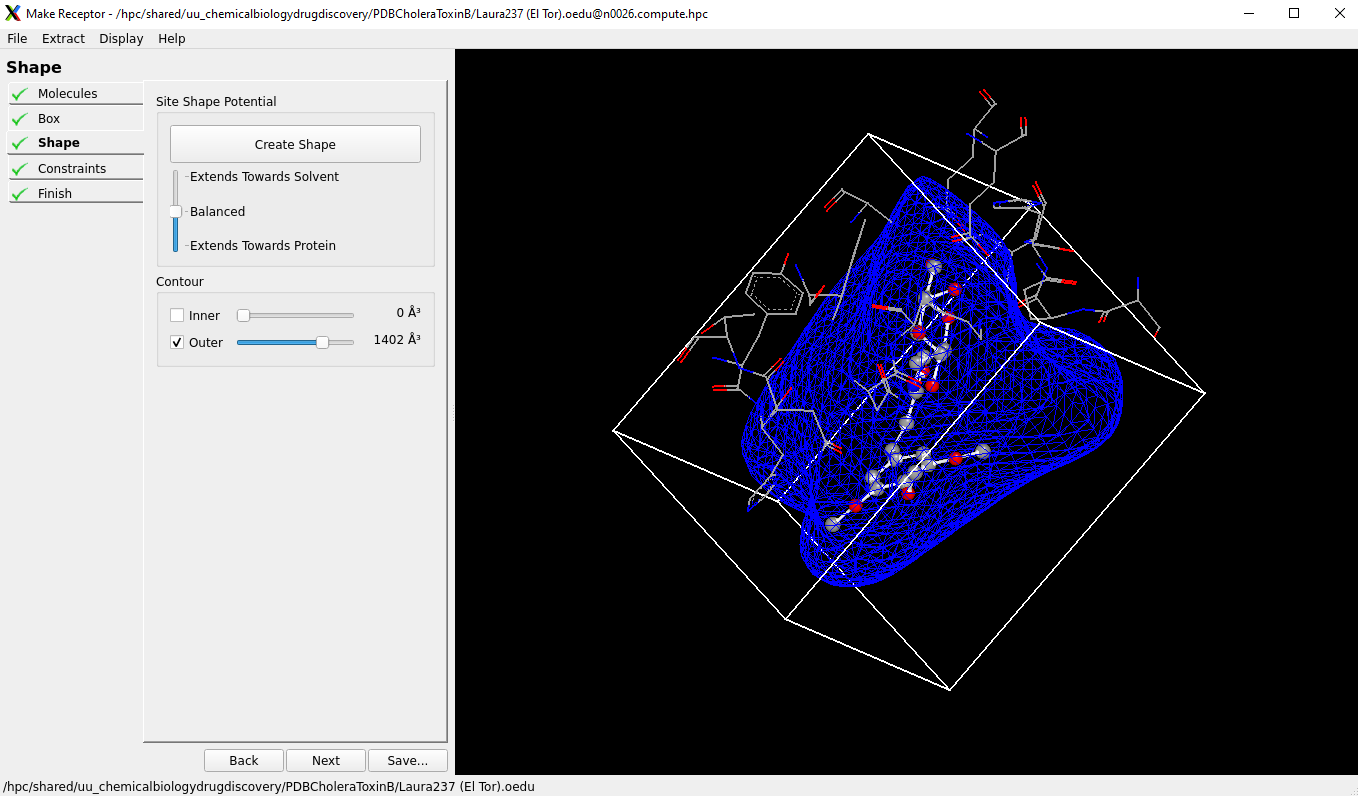
Bij dit voorbeeld twee MT’s (dit is Laura237 El Tor PDB) dit gold ook voor de NG5.oedu. De rest hadden allemaal HT’s. Bij allen is de bovenste gekozen.Afbeelding met grafiek

Automatisch gegenereerde beschrijving

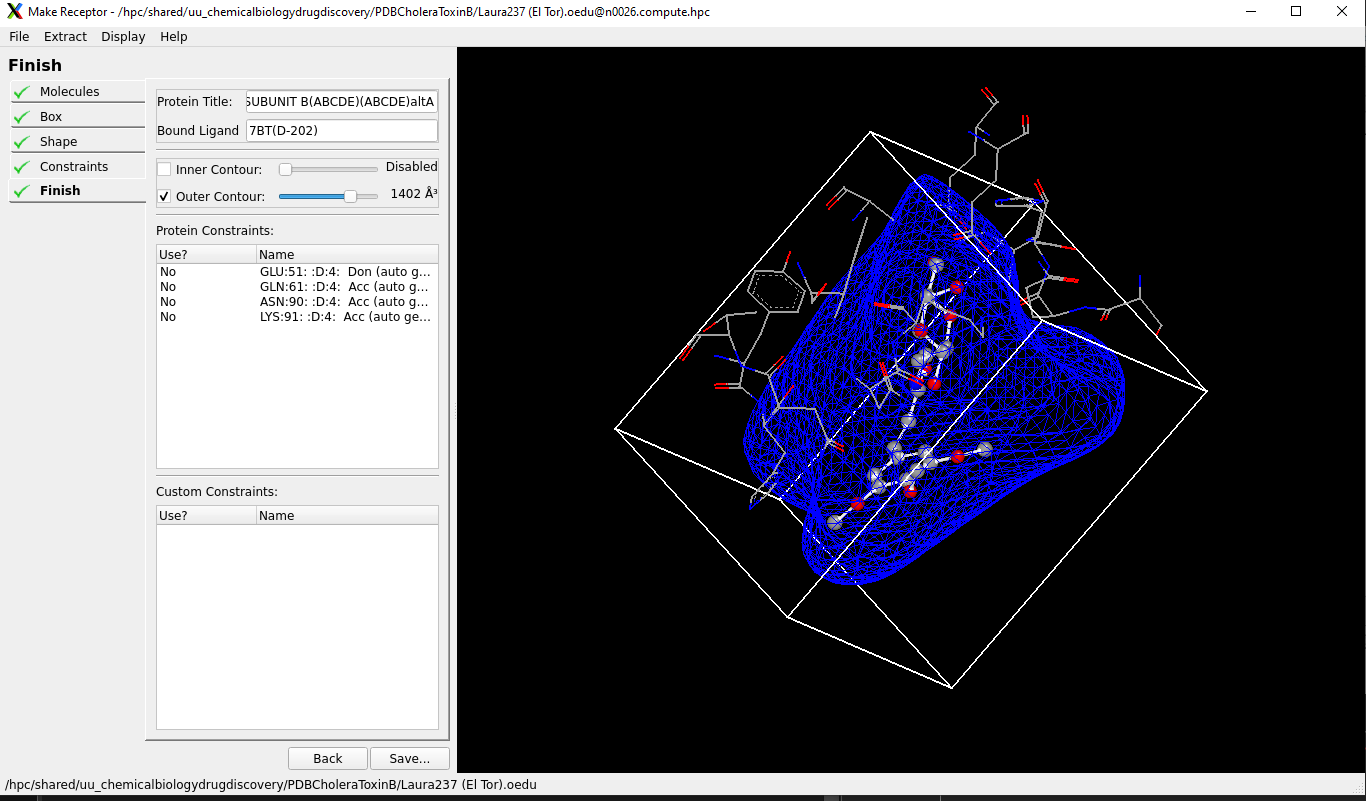
De box is kleiner gemaakt, goed passend om het ligand. Vervolgens is cavity detection gedaan met de molecular methode; deze is langzamer maar beter dan atomic.



Vervolgens is Create Shape gedaan met alles op default; balanced.



De constraints zijn zo gelaten (default) en bij Finish is het bestand opgeslaten tot een .oedu file.



Vrijdag 17 maart

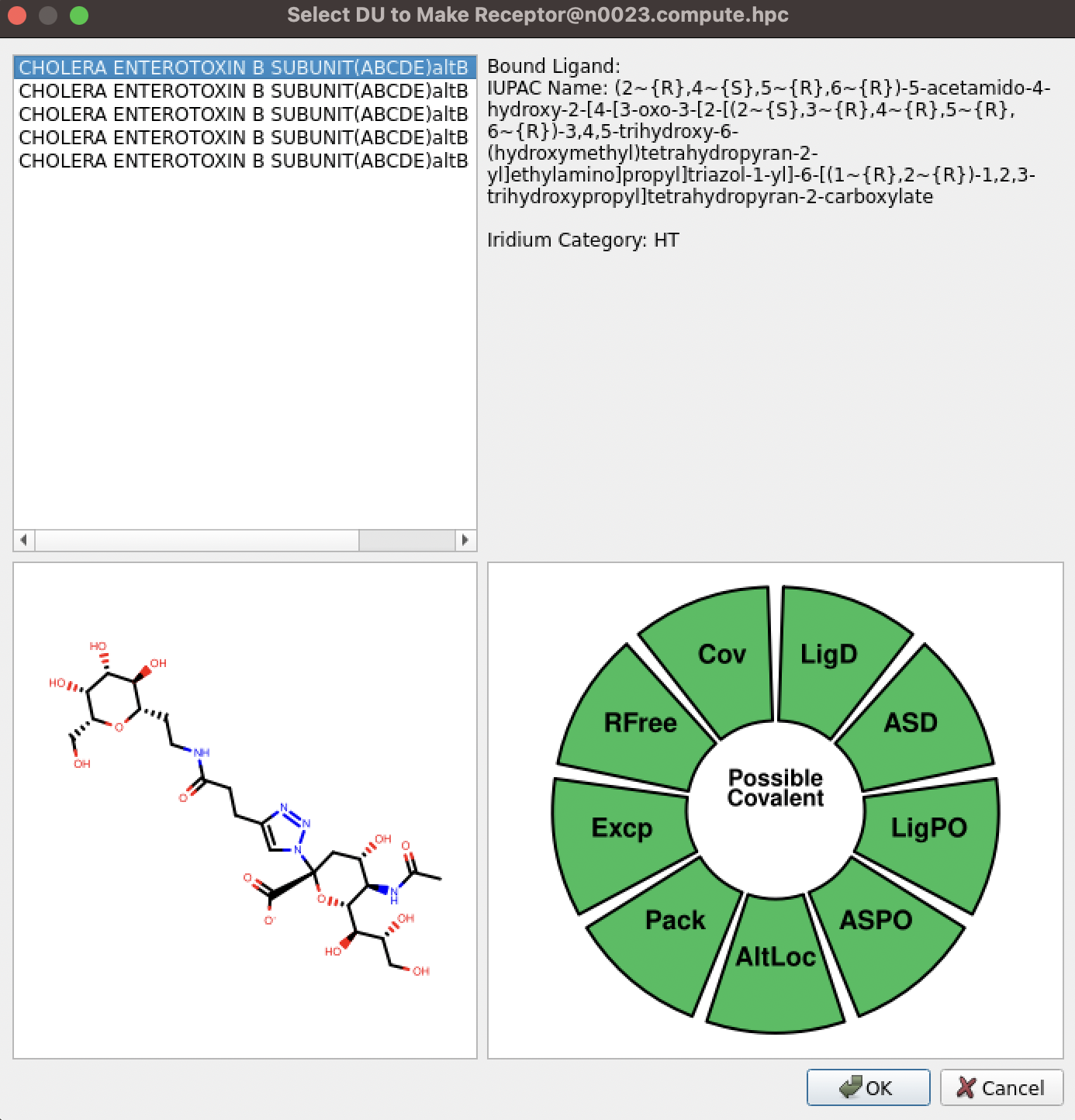
Rerun gedaan voor de 5lzh.pdb/PC262.oedu; deze was kwijt en gaf ook een error.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Hiervan is ook een logfile opgeslagen met alle details (staat op HPC in persoonlijke map) PATH invoeren

Wel gaf de .oedu HT waardes aan ondanks de foutmelding;



Screenshot gemaakt van GM1os in pocket;Afbeelding met kaart

Automatisch gegenereerde beschrijving

Script gerund voor uitrekenen RMSD’s met nieuwe .oedu:

python superposition.py --filenames NG5,PC262,GM1os,Laura237\_El\_Tor,MNPG --filetype oedu --proteinname CTB\_rerun

PATH resultaten

Voorbeeld resultaten

**Tuesday 21/03/2023**

Afbeelding met tekst

Automatisch gegenereerde beschrijving

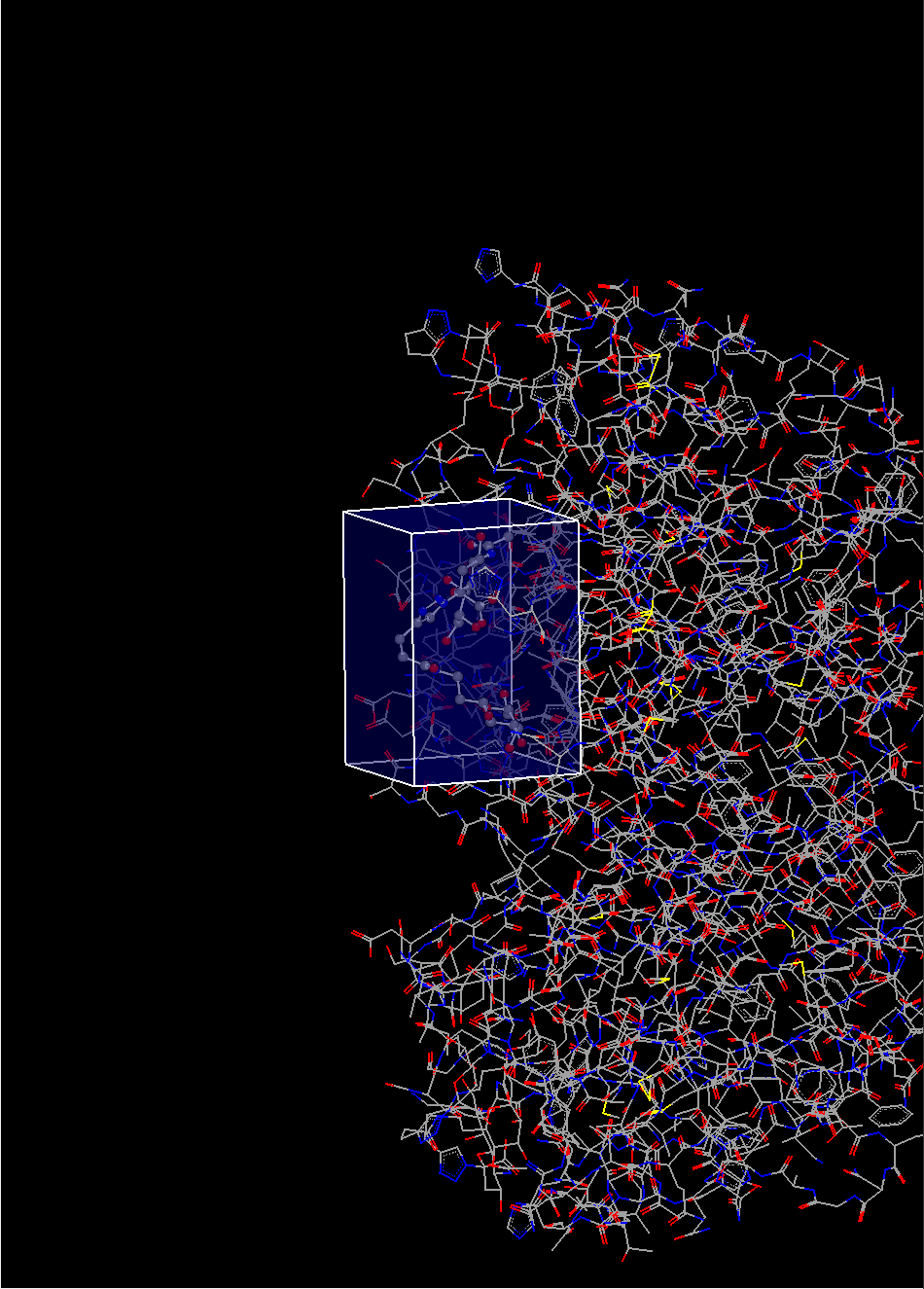
**1fgb.pdb is an empty cholera toxin B subunit (APO structure)**

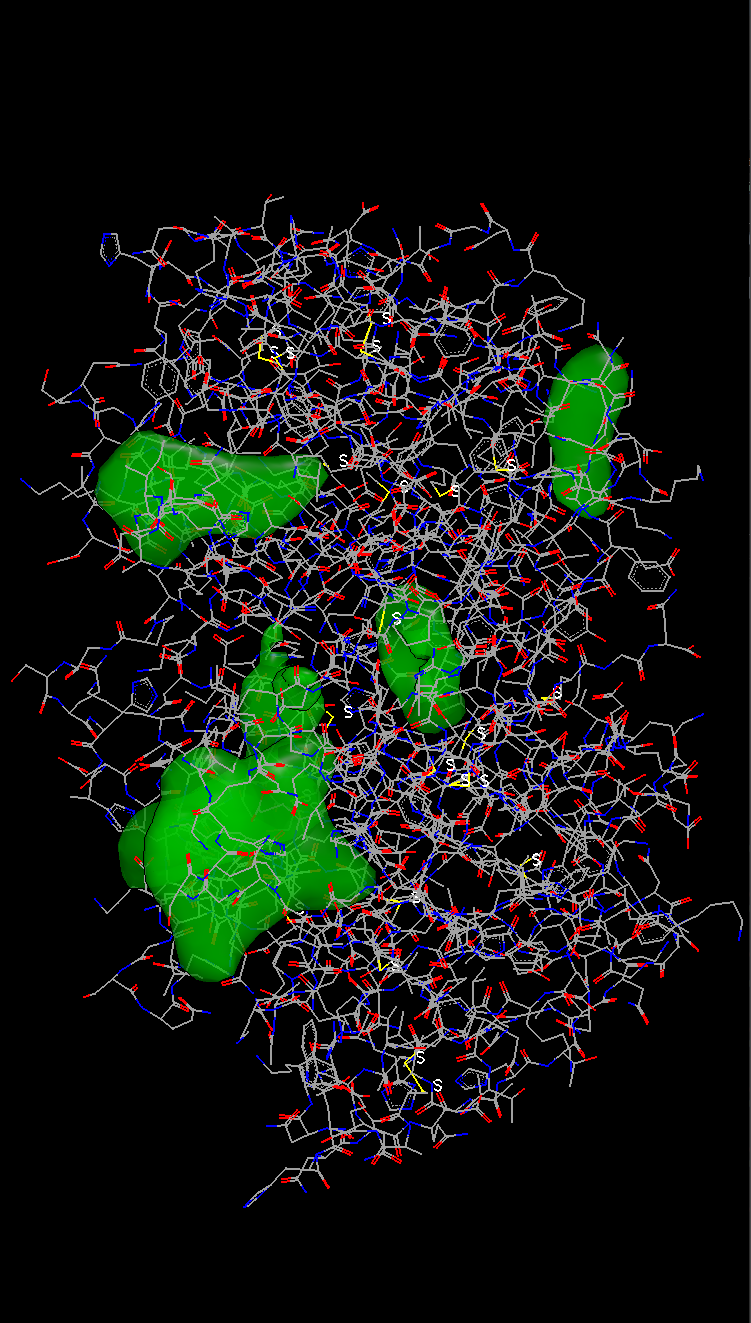
Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

APO structure using MakeReceptor. Load in, accept for apo structure. After that use molecular cavity detection. Select one of the green pockets; use a reference PDB file with bound ligand to find correct pocket/use literature. Further steps are the same for normal OEDU making. You can use a reference for faster site finding (at the beginning with file loading you can select a reference molecule). Cavities arent found correctly…

**VRAGEN WAT HET BESTE IS: KAN NIET DE JUISTE CAVITY VINDEN MET CAVITY DETECTION; WEL MET REFERENCE MOLECULE INLADEN IN HET BEGIN. Met sensitivitys pelen helpt met juiste vinden?**





**Monday 27/03/2023 and Tuesday 28/03/2023**

Ran spruce (command line version; easier to make choices/see metrics) and after that make receptor. This was done for the following PDB files: 3CHB, 1EEI, 5LZJ, 5LZG, 5LZH(nog niet gedaan; wacht op Keulen).

While ET CTB displays reduced affinity for blood group A and B antigens, both CT variants bind equally strongly to GM1. Therefore both were used.

First for every PDB the structure was retrieved using:

*getstructure pdbcode*

For every spruce run the following command was used (example with 5LZG):

*srun --time=03:00:00 --pty bash* #Start an interactive session for 3 hours (multiple spruce runs)

*spruce -in 5lzg.cif -map 5lzg.mtz -loop\_db\_filename ../rcsb\_spruce\_1\_0\_20200422.loop\_db* #Select input file, mapfile and loop db file

srun was used so any errors could be seen.

Below all screenshots from the output.log files for each pdb with the selected .oedu.

5LZG

Afbeelding met tekst

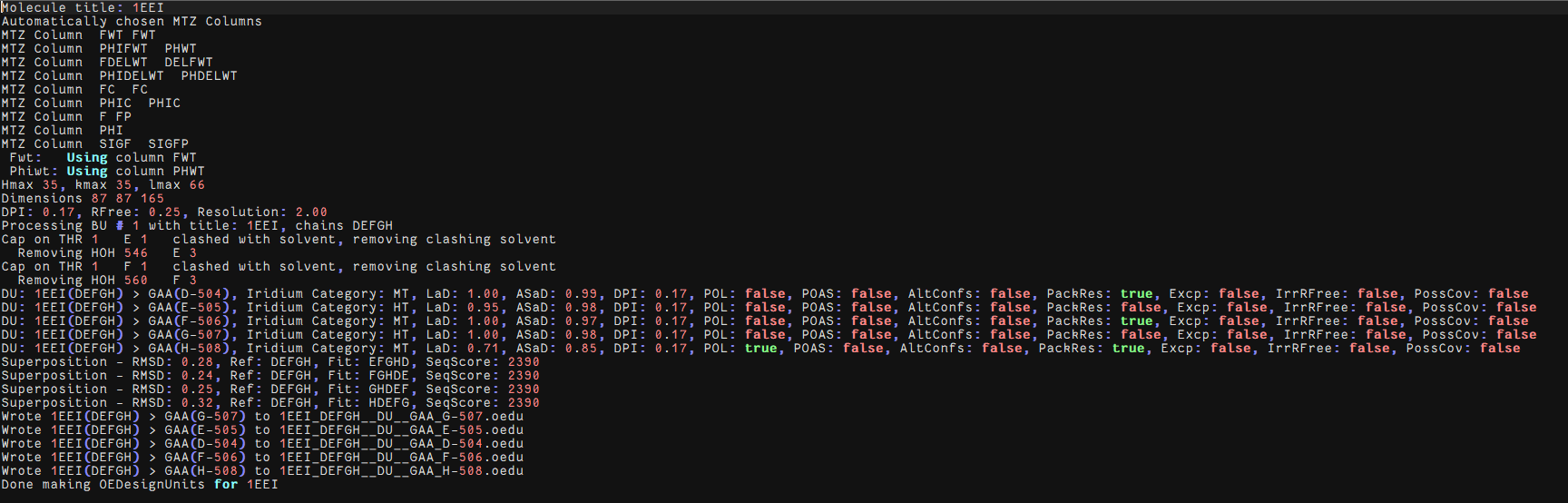
Automatisch gegenereerde beschrijving

B-201 is the only HT

Afbeelding met tekst

Automatisch gegenereerde beschrijving

1EEI



Two HT’s; only difference is LaD (density coverage of the ligand heavy atoms) score. Higher LaD is better (Warren et al., 2012), therefore DU 507 has been chosen.

5LZJ

Afbeelding met tekst

Automatisch gegenereerde beschrijving

D-202 is the best option due to a higher LaD score. Both DU’s fail at the PackRes filter step. This means that the precense of crystal packing residues near binding site is too low.

5LZH

Afbeelding met tekst

Automatisch gegenereerde beschrijving

Most likely wont be used: if used we use C-201 due to higher ASaD.

1PZJ

Afbeelding met tekst

Automatisch gegenereerde beschrijving

108 is chosen duet o highest LaD score, although packres is true. PackRes is described before. This is a MT structure, no HT’s were found.

1PZK

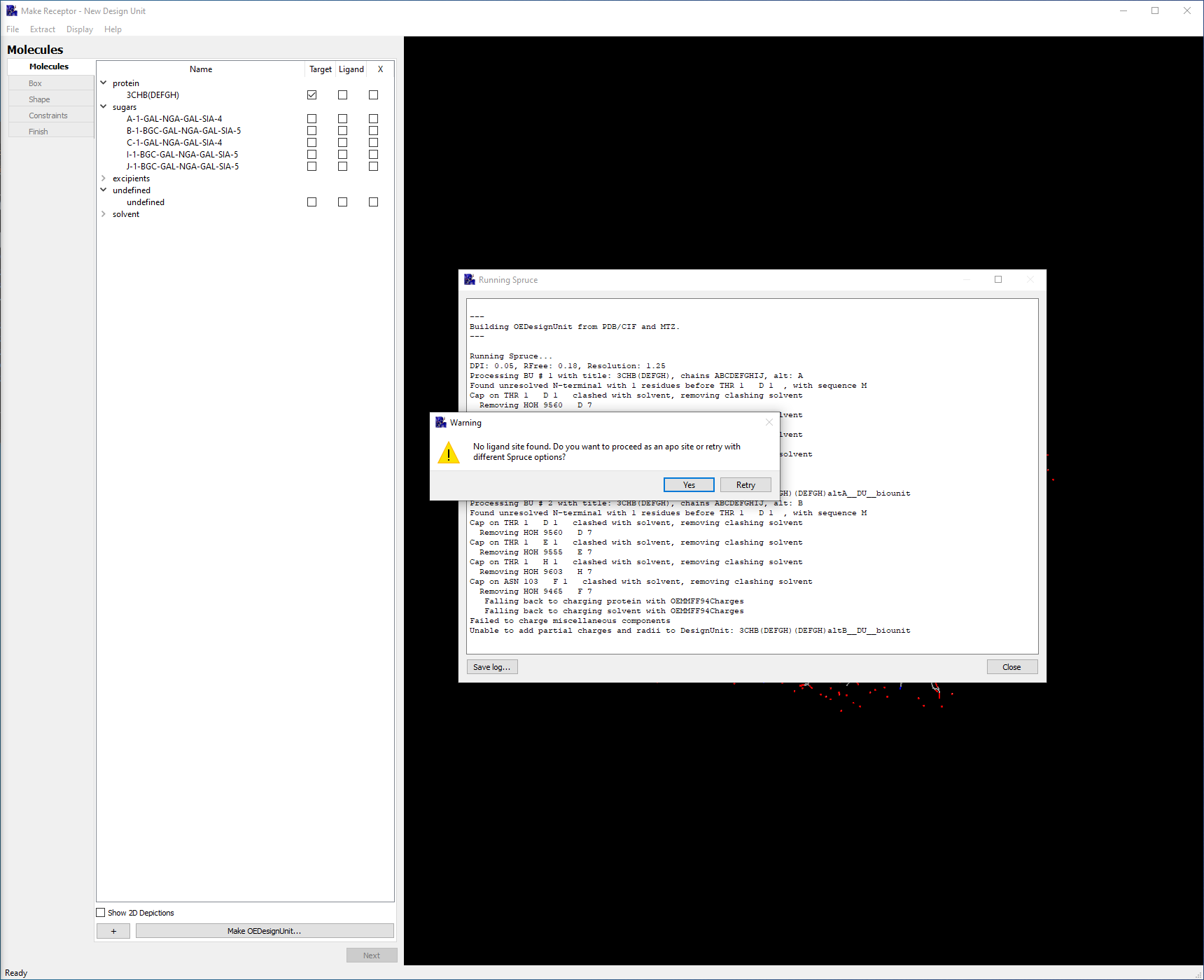
Afbeelding met tekst

Automatisch gegenereerde beschrijving

G-104 is chosen due to high LAD and ASaD scores. Only MT scores were given.

3CHB

This one is made with OEMakeReceptor as a ligand was not found. This was done by selecting the ligand in OEMakeReceptor as follows:

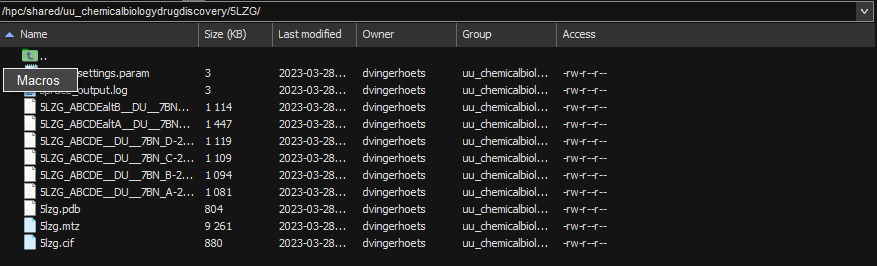


Here choose yes. And select a pentasacharide as ligand. This gives no iridium score? Achterwege laten.

Afbeelding met tekst

Automatisch gegenereerde beschrijving

All files were saved in corresponding directories in */hpc/shared/uu\_chemicalbiologydrugdiscovery/’PDBNAME’*



All chosen DU’s were put in MakeReceptor.

Make receptor was run with a GPU to speed up the visual aspects using the following command:

*srun --pty --x11 --time=03:00:00 makereceptor*

All .oedu files were loaded in using Open Design Unit/Receptor and checked.

Vanaf hier data niet meer genoemd… Alles staat er verder wel in!

**Molproperty graphs**

Molproperty graphs were made first by getting the molproperty table using the openeye script.

molproptable.py -in combined.sdf

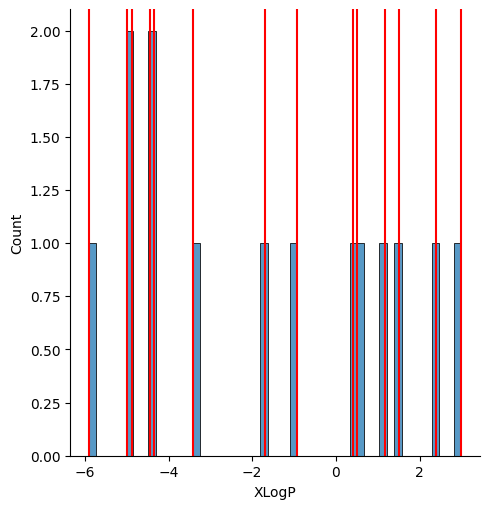
This wil give a molpropertytable from your whole database.

The following script was made to create graphs from this table:

import matplotlib.pyplot as plt  
import seaborn as sbn  
import dask.dataframe as dd  
  
actives = dd.read\_csv("actives.txt", header=0, sep='\t')  
database = dd.read\_csv('database.txt', header=0, sep='\t')  
properties = []  
for col in actives.columns:  
 properties.append(col)  
properties.remove('SMILES')  
properties.remove('Name')  
def propertyplots(properties):  
 for i in properties[160:180]:  
 plot = sbn.displot(database[i], bins=10)  
 plot.set(xlabel=i)  
 for j in actives[i]:  
 plt.axvline(j, color='red')  
 filename = i+'.png'  
 plot.savefig(filename)  
  
propertyplots(properties)

The script needs to be altered corresponding to the property you want to plot.

To plot it with actives you need to make a table for your actives and add the actives as seen in the script above.

Example graph of galectin run:

**RMSD array**

RMSD calculations between all PDB’s have been done in both PDB and oedu format using the superposition.py script, which combines all superposition methods from the superposition openeye tool:

import glob, os  
import statistics as st  
import numpy as np  
import pandas as pd  
import argparse  
  
parser = argparse.ArgumentParser()  
parser.add\_argument('--proteinname', type=str, required=True)  
parser.add\_argument('--filenames', type=str, required=True)  
parser.add\_argument('--filetype', type=str, required=True)  
args = parser.parse\_args()  
filetype = str('.')+args.filetype  
protein = args.proteinname  
pdblist = args.filenames.split(',')  
methods = ['global', 'site', 'ddm', 'sse', 'weighted', 'sitehopper']  
for m in methods:  
 if not os.path.exists(m):  
 os.makedirs(m)  
 if m == 'sse':  
 score = 'Tanimoto'  
 else:  
 score = 'RMSD'  
 rmsdlists = []  
 for i in pdblist:  
 rmsdlist = []  
 for j in pdblist:  
 os.system('superposition -ref '+str(i)+str(filetype)+' -fit '+str(j)+str(filetype)+' -prefix '+str(m)+'/'+str(i)+str(j)+' -method '+str(m)+' -log '+str(m)+'/'+str(i)+str(j)+'.log')  
 f = open(str(m)+'/'+str(i)+str(j)+'.log')  
 for line in f:  
 if score in line:  
 split\_line = line.split(' ')  
 rmsd = split\_line[3]  
 rmsd = rmsd.replace(',', '')  
 rmsdlist.append(float(rmsd))  
 rmsdlists.append(rmsdlist)  
 rmsdmatrix = np.stack(rmsdlists)  
 df = pd.DataFrame(rmsdmatrix, index=pdblist, columns=pdblist)  
 medians = []  
 means = []  
 for i in rmsdlists:  
 medians.append(st.median(i))  
 means.append(st.mean(i))  
 df['median'] = medians  
 df['mean'] = means  
 df.index.name = m  
 df.to\_csv(str(protein)+' '+str(m)+'.csv', index=True, header=True, sep=' ')  
  
writer = pd.ExcelWriter(str(protein)+' superposition results.xlsx')  
  
for filename in glob.glob(str(protein)+'\*.csv'):  
 usedmethod = filename.split('\t')  
 df\_csv = pd.read\_csv(filename, sep=' ')  
 (\_, f\_name) = os.path.split(filename)  
 (f\_shortname, \_) = os.path.splitext(f\_name)  
 df\_csv.to\_excel(writer, f\_shortname, index=False)  
  
writer.close()

Results where saved in a .xlsx file, containing matrices with all RMSD values.

**Selfdocking**

All actives were docked in its own .oedu. with default settings (Hybrid).

All actives were docked in all oedu’s resulting in only good scores for 1PZJ and 1EEI both were used for ROCS and HYBRID

**Correct naming of databases**

Databases have faulty naming, where OpenEye does give own names to molecules (molecule1, molecule2, molecule3… etc). To preserve the correct naming the following script has been used for the Enamine SDF files:

with open("Enamine\_CLOUD\_293cmpds\_20181010.sdf", "r") as f:  
 lines = f.readlines()  
with open("namefile.txt", "r") as nf:  
 nlines = nf.readlines()  
count = 0  
for i, j in enumerate(lines):  
 if "Mrv1813" in j:  
 if lines[i-1] == "\n":  
 lines[i-1] = nlines[count]  
 count += 1  
with open("Enamine\_CLOUD.sdf", "w") as wf:  
 for i in lines:  
 wf.write(i)

The file and “Mrv1813” need to be altered accordingly.

Another way of adding names to SDF format which is easier is to use the tool openbabel. The following command was used in a srun session:

obabel infile.sdf -osdf outfile.sdf –append “Catalog ID”

The option append adds a SDF property to the title of the molecule. Open the SDF file using a text tool to look at which property contains the ID you want to title the molecule with.

**Splitting the MCULE database**

The MCULE database is in .smi format. Splitting .smi format can easily done by splitting on amount of lines:

split -l3000000 infile.smi

This command creates files each containing 3.000.000 lines and therefore molecules. This was done for the MCULE database resulting in 14 smi files.

**Filtering the MCULE database**

The MCULE database was filtered using sbatch and the openeye filter application:

sbatch –time=05:00:00 –wrap=“filter -in ../mcule\_split\_1.smi -out mcule\_Filter\_1.smi -prefix mcule\_1 -sdtag true”

This was done for all 14 splitfiles.

**FLIPPER**

Flipper was ran for all databases uising sbatch and the following settings:

sbatch –time=20:00:00 –mem=30G –wrap=”flipper -in ../Filter/mcule\_Filter\_1.smi -out mcule\_Filter\_Flipper\_1.smi -prefix mcule\_1 -progress percent -warts true”

This was done for all Enamine databases and mcule splits.

**Tautomers**

Flipper was ran for all databases using sbatch and the following settings:

Sbatch –time=20:00:00 –mem=30G –wrap=”tautomers -in ../Flipper/mcule\_Filter\_Flipper\_1.smi -out mcule\_Filter\_Flipper\_Tautomers\_1.smi -prefix mcule\_1 -warts true”

This was done for all Enamine flipped databases and mcule flipped splits.

**OMEGA**

Omega has been done in both omega\_rocs and omega\_pose as they ask for different amount of conformers. This was done using sbatch and the following command:

Sbatch –time=30:00:00 –mem-per-cpu=3G -c 30 –wrap=”omega pose -mpi\_np 30 -in ../Tautomers/mcule\_Filter\_Flipper\_Tautomers\_1.smi -out mcule\_pose\_1.oeb.gz -prefix mcule\_pose\_1 -progress percent”

This was done with both omega rocs and pose for all databases.

**OMEGA splitting**

After finding out runs were still too slow with Hybrid OMEGA MCULE pose databases were further split using the following molchunk.py code:

#!/usr/bin/env python  
# (C) 2022 Cadence Design Systems, Inc. (Cadence)   
# All rights reserved.  
# TERMS FOR USE OF SAMPLE CODE The software below ("Sample Code") is  
# provided to current licensees or subscribers of Cadence products or  
# SaaS offerings (each a "Customer").  
# Customer is hereby permitted to use, copy, and modify the Sample Code,  
# subject to these terms. Cadence claims no rights to Customer's  
# modifications. Modification of Sample Code is at Customer's sole and  
# exclusive risk. Sample Code may require Customer to have a then  
# current license or subscription to the applicable Cadence offering.  
# THE SAMPLE CODE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,  
# EXPRESS OR IMPLIED. OPENEYE DISCLAIMS ALL WARRANTIES, INCLUDING, BUT  
# NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY, FITNESS FOR A  
# PARTICULAR PURPOSE AND NONINFRINGEMENT. In no event shall Cadence be  
# liable for any damages or liability in connection with the Sample Code  
# or its use.  
  
#############################################################################  
# Split molecule file into N chunks or chunks of size N  
#############################################################################  
import os  
import sys  
from openeye import oechem  
  
  
def NewOutputStream(outbase, ext, chunk):  
 newname = outbase + ('\_%07d' % chunk) + ext  
 ofs = oechem.oemolostream()  
 if not ofs.open(newname):  
 oechem.OEThrow.Fatal("Unable to open %s for writing" % newname)  
 return ofs  
  
  
def SplitNParts(ifs, nparts, countconfs, outbase, ext):  
 # first read entire file to determine number of molecules  
 molconfcount = 0  
 for mol in ifs.GetOEMols():  
 if countconfs:  
 molconfcount += mol.NumConfs()  
 else:  
 molconfcount += 1  
 ifs.rewind()  
  
 chunksize, lft = divmod(molconfcount, nparts)  
 if lft != 0:  
 chunksize += 1  
 chunk = 1  
 count = 0  
  
 ofs = NewOutputStream(outbase, ext, chunk)  
 for mol in ifs.GetOEMols():  
 if countconfs:  
 count += mol.NumConfs()  
 else:  
 count += 1  
 if count > chunksize:  
 if chunk == lft:  
 chunksize -= 1  
  
 ofs.close()  
 chunk += 1  
 count = 1  
 ofs = NewOutputStream(outbase, ext, chunk)  
  
 oechem.OEWriteMolecule(ofs, mol)  
  
  
def SplitChunk(ifs, chunksize, countconfs, outbase, ext):  
 chunk = 1  
 ofs = NewOutputStream(outbase, ext, chunk)  
  
 count = 0  
 for mol in ifs.GetOEMols():  
 if count >= chunksize:  
 ofs.close()  
 count = 0  
 chunk += 1  
 ofs = NewOutputStream(outbase, ext, chunk)  
  
 if countconfs:  
 count += mol.NumConfs()  
 else:  
 count += 1  
 oechem.OEWriteMolecule(ofs, mol)  
  
  
def main(argv=[\_\_name\_\_]):  
 itf = oechem.OEInterface(InterfaceData, argv)  
  
 if not (itf.HasInt("-num") ^ itf.HasInt("-size")):  
 oechem.OEThrow.Fatal("Number of chunks (-num) or the size of each chunk "  
 "(-size) must be specified and are mutually exclusive.")  
  
 ifs = oechem.oemolistream()  
 oechem.OEPreserveRotCompress(ifs) # ADDED TO PRESERVE COMPRESSION  
 if not ifs.open(itf.GetString("-i")):  
 oechem.OEThrow.Fatal("Unable to open %s for reading" % itf.GetString("-i"))  
  
 if (ifs.GetFormat() != oechem.OEFormat\_OEB):  
 ifs.SetConfTest(oechem.OEIsomericConfTest(False))  
 outbase, ext = os.path.splitext(itf.GetString("-o"))  
  
 if ext == '':  
 oechem.OEThrow.Fatal("Failed to find file extension")  
  
 if ext == '.gz':  
 outbase, ext = os.path.splitext(outbase)  
 ext = ext + '.gz'  
  
 countconfs = itf.GetBool("-confs")  
  
 if itf.HasInt("-num"):  
 nparts = itf.GetInt("-num")  
 SplitNParts(ifs, nparts, countconfs, outbase, ext)  
 else:  
 chunksize = itf.GetInt("-size")  
 SplitChunk(ifs, chunksize, countconfs, outbase, ext)  
  
#############################################################################  
  
  
InterfaceData = """\  
!BRIEF -num|-size [-i] <input> [-o] <output>  
!PARAMETER -i 1  
 !TYPE string  
 !REQUIRED true  
 !BRIEF Input file name  
 !KEYLESS 1  
!END  
!PARAMETER -o 2  
 !TYPE string  
 !REQUIRED true  
 !BRIEF Output file name  
 !KEYLESS 2  
!END  
!PARAMETER -num 3  
 !TYPE int  
 !BRIEF The number of chunks  
!END  
!PARAMETER -size 4  
 !TYPE int  
 !BRIEF The size of each chunk  
!END  
!PARAMETER -confs 5  
 !TYPE bool  
 !DEFAULT true  
 !BRIEF Split by number of conformers not molecules  
!END  
"""  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 sys.exit(main(sys.argv))

This code was used as follows:

python molchunk.py -confs false -size 400000 -i input.oeb.gz -o output.oeb.gz

This exact code needs to be used as it keeps OMEGA compression!

**Constraints 1PZJ**

For 1PZJ constrains were set using MakeReceptor:

Make receptor and other visual tools can be opened using srun.

srun –pty -x11 –mem=30G makereceptor

Afbeelding met tekst, schermopname, kaart, diagram

Automatisch gegenereerde beschrijving

**HYBRID runs**

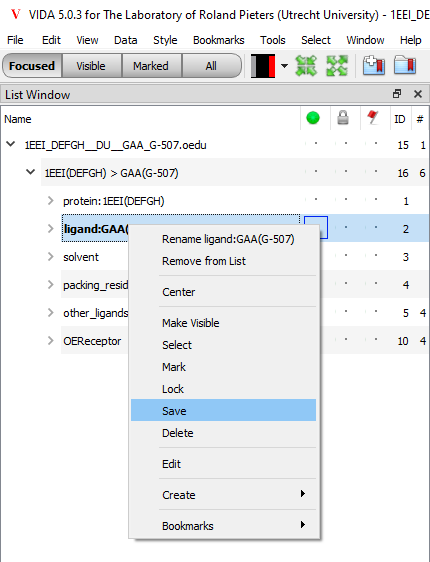
HYBRID was ran on all tautomers&flipper databases using the following sbatch command:

sbatch –time=30:00:00 -c 80 –mem-per-cpu=3G -o slurm\_%x.out –job\_name=”1EEI\_hybrid\_1” –wrap=”hybrid -mpi\_np 160 -receptor receptor.oedu -dbase ../Hybrid\_run\_chunks/chunk1\_\_0000001.oeb.gz -hitlist\_size 0 -prefix 1EEI\_hybrid\_Mcule\_1\_1 ”

%x is for naming the slurm output files with job name

**Ligand extraction from .oedu**

Ligands were extracted using VIDA for both 1PZJ and 1EEI:

****

After this both ligands were opened in VIDA and saved together in a ligands.oeb file containing both ligands as ROC can run with multiple queries at a time.

**Translating Enamine databases from SDF to SMILES**

Done with openbabel

Obabel –i enamine.sdf -osmi enamine.smi

**ROCS runs**

With ROCS both 1EEI and 1PZJ queries could be ran in one ROCS run. Databases were saved in a list file, which makes it possible to run multiple databases in 1 rocs run:

Afbeelding met tekst, schermopname

Automatisch gegenereerde beschrijving

Sbatch –time=40:00:00 -c 80 –mem-per-cpu=3G -o slurm\_%x.out –job\_name=”ROCS\_1EEI\_1PZJ\_” -wrap=”rocs -mpi\_np 160 -query ligands.oeb -dbase databases.list -prefix ROCS\_CTB -besthits 0 -verbose true”

**Data extraction & reciprocal calculation**

Scores where extracted from HYBRID scoring list by first combining al scoring lists and only saving columns containing molecule names and Chemgauss scoring, this was done with awk:

awk “{print $1, $2}” \*score.txt > combined\_scores.txt

After this the file was sorted on the second column containg the chemgauss scores:

sort -k2 -n combined\_scores.txt > combined\_sorted.txt

After this the warts were removed using sed.

sed -i -e “s/\_.\* / /” combined\_sorted.txt

Now, the duplicates are removed using awk so we only are left with the best scores per molecule.

awk “!seen[$1]++” combined\_sorted.txt > combined\_nodups.txt

This was done also for ROCS using the same commands, only selecting the tanimoto score with the score extracting instead of chemgauss.

Having now scoring lists, with nodups and sorted. The reciprocal is calculated as following using awk:

awk “{print $1, 1/NR}” combined\_nodups.txt > reciprocal.txt

This was done for both 1PZJ, 1EEI and both hybrid en rocs scoring lists.

For calculating the Reciprocal\_Sum the following python script was made and used:

hybrid\_dictionary = {}  
with open("D:/reciprocal\_hybrid\_1EEI.txt") as f:  
 for line in f:  
 (key, val) = line.split()  
 hybrid\_dictionary[key] = val  
  
rocs\_dictionary = {}  
with open("D:/ROCS\_1EEI\_reciprocal.txt") as f:  
 for line in f:  
 (key, val) = line.split()  
 rocs\_dictionary[key] = val  
  
hybrid\_mols = list(hybrid\_dictionary.keys())  
rocs\_mols = list(rocs\_dictionary)  
mols\_both = set(hybrid\_mols) & set(rocs\_mols)  
  
reciprocal\_sum = []  
for i in mols\_both:  
 reciprocal\_sum.append(i+' '+hybrid\_dictionary[i]+' '+rocs\_dictionary[i]+' '+str((float(hybrid\_dictionary[i])+float(rocs\_dictionary[i]))))  
  
with open('reciprocal\_sum\_1EEI.txt', 'w') as fp:  
 fp.write('\n'.join(reciprocal\_sum))

After getting the reciprocal sum lists, these can be sorted again using sort:

sort -k4 -gr reciprocal\_sum.txt

This is done for both 1PZJ and 1EEI.

**Extracting 3D docked structures**

Top1000 molecules were selected from the sorted reciprocals using the following command:

awk “{print $1} NR==1000{exit}” reciprocal.txt > top1000mols.txt

Now we have a top1000 mols list, which we can select in the docked.oeb.gz files. However these had warts and we did not so we extracted the titles from the docked files using the following command:

gettitles docked.oeb.gz > titlesdocked.txt

This was done for all docked oeb.gz files.

Now we can select the molecules with warts using grep:

grep -F -f top1000mols.txt titlesdocked.txt > extractmols.txt

We can extract this mols from the docked oeb.gz using molextract:

molextract -list extractmols.txt docked.oeb.gz select.oeb.gz

This select.oeb.gz was viewed in VIDA together with the corresponding design unit.