Crystals **2015**, *5*, 617-633; doi:10.3390/cryst5040617

OPEN ACCESS

crystals

ISSN 2073-4352

www.mdpi.com/journal/crystals

Supplementary Information

The Search for Hydrogen Stores on a Large Scale; A Straightforward and Automated Open Database Analysis as a First Sweep for Candidate Materials

Joachim Breternitz and Duncan H. Gregory *

WestCHEM, School of Chemistry, University of Glasgow, University Avenue, Glasgow, G12 8QQ, UK; E-Mails: joachim@chem.gla.ac.uk; Duncan.Gregory@glasgow.ac.uk

* Author to whom correspondence should be addressed; E-Mail: Duncan.Gregory@glasgow.ac.uk; Tel.: +44 141 330 6438.

This supporting information can be best understood as technical guidelines for the attached programs and explains the functionality of the programs and how to run them.

S1. Data Harvesting Procedure

A copy of the COD was obtained using the method described on the homepage (http://wiki.crystallography.net/howtoobtaincod/) and treated with the python code written below. In order to run the code, an index file ("files") is needed, which contains the name of the cif files with their absolute or relative location. Such a file can be obtained in a Linux system (and probably other Unixoid systems) by running the following command in the top-level folder of the local COD copy:

```
ls -R cod/cif | awk '
/:$/&&f{s=$0;f=0}
/:$/&&!f{sub(/:$/,"");s=$0;f=1;next}
NF&&f{ print s"/"$0 }' > files
```

With this "files"-file, the data harvesting procedure can be started using the following python script in the same folder (comments are given in green). In the command line, this can be achieved by typing "python harvester.py". Under windows, a simple double click may be enough (provided python is installed).

```
import CifFile   #Import PyCifRW
import re       # Import regular expression module
list=open('files') # Open input file containing the cif file directions
out1=open('capacity', 'w') # Open the export files
out2=open('formulae','w')
```

```
Molweights={'H':1.008,'D':1.008, 'He':4.002, 'Li':6.94, 'Be':9.012, 'B':10.81, 'C':12.01,
'N':14.01, 'O':16.00, 'F':19.00, 'Ne':20.18, 'Na':22.99, 'Mg':24.31, 'Al':26.98,
'Si':28.09, 'P':30.97, 'S':32.06, 'Cl':35.45, 'Ar':39.95, 'K':39.10, 'Ca':40.08, 'Sc':44.96,
'Ti':47.87, 'V':50.94, 'Cr':52.00, 'Mn':54.94, 'Fe':55.85, 'Co':58.93, 'Ni':58.69,
'Cu':63.55, 'Zn':65.38, 'Ga':69.72, 'Ge':72.63, 'As':74.92, 'Se':78.97, 'Br':79.90,
'Kr':83.80, 'Rb':85.47, 'Sr':87.62, 'Y':88.91, 'Zr':91.22, 'Nb':92.11, 'Mo':95.95,
'Tc':97.91, 'Ru':101.07, 'Rh':102.91, 'Pd':106.42, 'Ag':107.87, 'Cd':112.41, 'In':114.82,
'Sn':118.71, 'Sb':121.76, 'Te':127.60, 'I':126.90, 'Xe':131.29, 'Cs':132.91, 'Ba':137.33,
'La':138.91, 'Ce':140.12, 'Pr':140.91, 'Nd':144.24, 'Pm':144.91, 'Sm':150.36,
'Eu':151.96, 'Gd':157.25, 'Tb':158.93, 'Dy':162.50, 'Ho':164.93, 'Er':167.26,
'Tm':168.93, 'Yb':173.05, 'Lu':174.97, 'Hf':178.49, 'Ta':180.95, 'W':183.84,
'Re':186.21, 'Os':190.23, 'Ir':192.22, 'Pt':195.08, 'Au':196.97, 'Hg':200.59, 'Tl':204.38,
'Pb':207.2, 'Bi':208.98, 'Po':208.98, 'At':209.99, 'Rn':222.02, 'Ac':227.03, 'Th':232.04,
'Pa':231.04, 'U':238.03, 'Np':237.05, 'Pu':244.06, 'Am':243.06, 'Cm':247.07,
'Bk':247.07, 'Cf':251.08}

# the hardcoded molecular weights for the elements avoids wrongly given molecular
weights and ambiguities with different Z.
# Also it allows the direct treatment of D-containing species as the molecular mass is
set to that of H.

for line in list: # Iteration over all lists in the files input

..try: # Exception handling to avoid the break-up of the program if one cif file is
erroneous. No output is created for these.
….if line[-5:-1] != '.cif': # Test if file is a cif-file
……..continue
….Molweight=0 # Reset the Molweight variable
….Name=line
….try: # Error handling if cif file cannot be read (i.e. due to wrong syntax)
…..cif=CifFile.ReadCif(Name)

….except:

…..continue
….block=re.findall(r'\d+',line) # extraction of block name for the cif file (the COD
number in the filename)

….print block[-1]
….datablock=block[-1]

….try: # Extraction of the Information needed. If one is not given, the cif-file is not
taken into account (via error handling)
…..CODnumber=cif[datablock]['_cod_database_code']
…..SumFormula=cif[datablock]['_chemical_formula_sum']
```

```
…..Density=cif[datablock]['_exptl_crystal_density_diffrn']
….except:
…..continue
….if Density == '?': # Omit cif files where the density is given as “?”
…..continue
….SumFormula.replace('\n',' ').replace('\r','') # Long Sum formulae contain line breaks that will
….#falsify the line numbers between the two output files. Therefore, the linebreak signs are stripped.
….elem=re.findall(r'[a-zA-Z]+',SumFormula) # Get the elements from the sum formula
….boolean = 'H' in elem or 'D' in elem
….if boolean == False: # Omit non H or D containing cifs
……..continue
….elements=re.findall(r'[a-zA-Z0-9.]+',SumFormula) # Get Elements with counts from Sum formula
….for item in elements: # Calculation of the molecular mass
……..elementscount=re.findall(r'[0-9.]+',item)
……..numbi=elements.index(item)
……..element=re.findall(r'[a-zA-Z]+',item)[0]
……..elements[numbi]=element
……..if not elementscount:
……….elementscount.append(1)
……..if element == 'H' or element == 'D':
……….Hcount=elementscount[0] # Fetching number of H/D atoms for gravimetric capacity
……..boolean = element in Molweights
……..if boolean == False:
……….continue
……..Molweight = Molweight + Molweights[element]*float(elementscount[0])
….gravcapacity=float(Hcount)*1.008*1000/Molweight # Calculation of the gravimetric capacity (in mg/g)
….Dens=re.findall(r'[0-9.]+',Density) # Fetching the density
….volcapacity=gravcapacity*float(Dens[0]) # Calculation of the volumetric capacity (in g/L)
….print1 = str(CODnumber) +' '+ str(gravcapacity) +' '+ str(volcapacity) + '\n' # Produce output
….out1.write(print1)
….print2 = str(CODnumber) +' +'+ str(SumFormula) + '+\n'
….out2.write(print2)
..except:
….continue
```

```
list.close()
```

This produces the two main output files “capacity” and “formulae”. These need to be transferred into the folder of the processing program.

S2. Data Processing Procedure

The GUI of the data analyser is largely self-explanatory. It takes the information as asked and passes it to the searching routine. This will write to two new output files prefix-capacity and prefix-formulae, which contain only the structures that fall into the requested subsets. Please note that some fields must not be left blank, but these contain default values that reach over the whole range.

In the element symbols lists, any spacer except for letters can be used.

```
from Tkinter import * #import Tkinter module
import re #import regular expressions module
def execution(): # definition of the sorting routine

….Result.config(text='working') # print status in GUI
….main.update_idletasks()
….yes = Elemlist.get() # get mandatory elements
….no = NoElemlist.get() # get forbidden elements
….nummax = int(NumElMax.get()) # get maximum number of elements
….nummin = int(NumElMin.get()) # get minimum number of elements
….outname=Prefix.get() #get name prefix
….gravminimal=float(gravmin.get()) # get minimal gravimetric capacity
….gravmaximal=float(gravmax.get()) # get maximal gravimetric capacity
….volminimal=float(volmin.get()) # get minimal volumetric capacity
….volmaximal=float(volmax.get()) # get maximal volumetric capacity
….outcapacity = outname + '-capacity' # define outfiles
….outformulae = outname + '-formulae'
….outc = open(outcapacity, 'w')
….outf = open(outformulae, 'w')
….yeslist = re.findall('[A-Za-z]+', yes) #get list of mandatory elements
….nolist = re.findall('[A-Za-z]+', no) #get list of forbidden elements
….formulae = open('formulae')
….capacity = open ('capacity')
….for line in formulae: # iterate over all lines in formulae
……..linecap = capacity.readline()
……..capacitylist = re.findall('[0-9.]+', linecap)
……..formulaelist = re.findall('[A-Za-z]+', line)
……..yesnumber = len(set(yeslist).intersection(formulaelist))
……..nonumber = len(set(nolist).intersection(formulaelist))
```

```
……..if yesnumber == len(yeslist) and nonumber == 0 and len(formulaelist) <=
nummax and len(formulaelist) >= nummin and     float(capacitylist[1]) >=
gravminimal and float(capacitylist[1]) <= gravmaximal and float(capacitylist[2]) >=
volminimal and     float(capacitylist[2]) <= volmaximal:
……#compare line with given values and print those that fit.
……outc.write(linecap)
……outf.write(line)
….Result.config(text='Done!')

main=Tk()
main.wm_title('COD database analysis')

#Definition of the GUI and setup of the window.

Label(main,text='Hydrogen Storage Materials from COD', font='Helvetica 16
bold').grid(row=0, columnspan=3)
Label(main,text='\nThis programme was written at the University of Glasgow\nin the
group of Prof. Duncan H. Gregory by Joachim     Breternitz.\nIt is free for use for the
academic community.\n\n').grid(row=1, columnspan=3)

LabelName=Label(main,text='Please name the prefix \nfor the output files:\n
\n').grid(row=2, sticky=E)
LabelEl=Label(main,text='Please type the element symbols\n that must be present:\n
\n').grid(row=3, sticky=E)
LabelNoEL=Label(main,text='Please type the element symbols\n that must NOT be
present:\n \n').grid(row=4, sticky=E)
LabelNumEl=Label(main,text='Please give the minimum and maximum\n number of
elements:\n \n').grid(row=5, sticky=E)
Labelgrav=Label(main,text='Please give the minimum and maximum\n levels of the
gravimetric capacity (mg(H2)/g):\n \n').grid(row=6,     sticky=E)
Labelvol=Label(main,text='Please give the minimum and maximum\n levels of the
volumetric capacity (g(H2)/L):\n \n').grid(row=7,     sticky=E)

Prefix=Entry(main)
Prefix.grid(row=2, column=1,columnspan=2, sticky=N)

Elemlist=Entry(main)
Elemlist.grid(row=3, column=1,columnspan=2, sticky=N)
NoElemlist=Entry(main)
NoElemlist.grid(row=4, column=1,columnspan=2, sticky=N)

NumElMax=Entry(main,  width=9)
NumElMax.insert(0, '99')
NumElMax.grid(row=5, column=2, sticky=N)
NumElMin=Entry(main, width=9)
```

```
NumElMin.insert(0, '1')
NumElMin.grid(row=5, column=1, sticky=N)

gravmin=Entry(main, width=9)
gravmin.insert(0, '0')
gravmin.grid(row=6, column=1, sticky=N)
gravmax=Entry(main, width=9)
gravmax.insert(0, '1000')
gravmax.grid(row=6, column=2, sticky=N)

volmin=Entry(main, width=9)
volmin.insert(0, '0')
volmin.grid(row=7, column=1, sticky=N)
volmax=Entry(main, width=9)
volmax.insert(0, '1000')
volmax.grid(row=7, column=2, sticky=N)

Buttonexec=Button(main,text='Search',command=execution)
Buttonexec.grid(row=8, columnspan=3)

Result=Label(main,text='        ')
Result.grid(row=9,columnspan=3)

main.mainloop()
```

The capacity file can be easily plotted using a program of preference. A simple option is gnuplot, in which the output can be produced using the following command:

```
plot ‘prefix-capacity’ u 2:3
```

The output file can also be imported to spreadsheet programs as a delimited text file, defining a space as the delimiter.