Computational Materials Repository

Landis, David Dominic

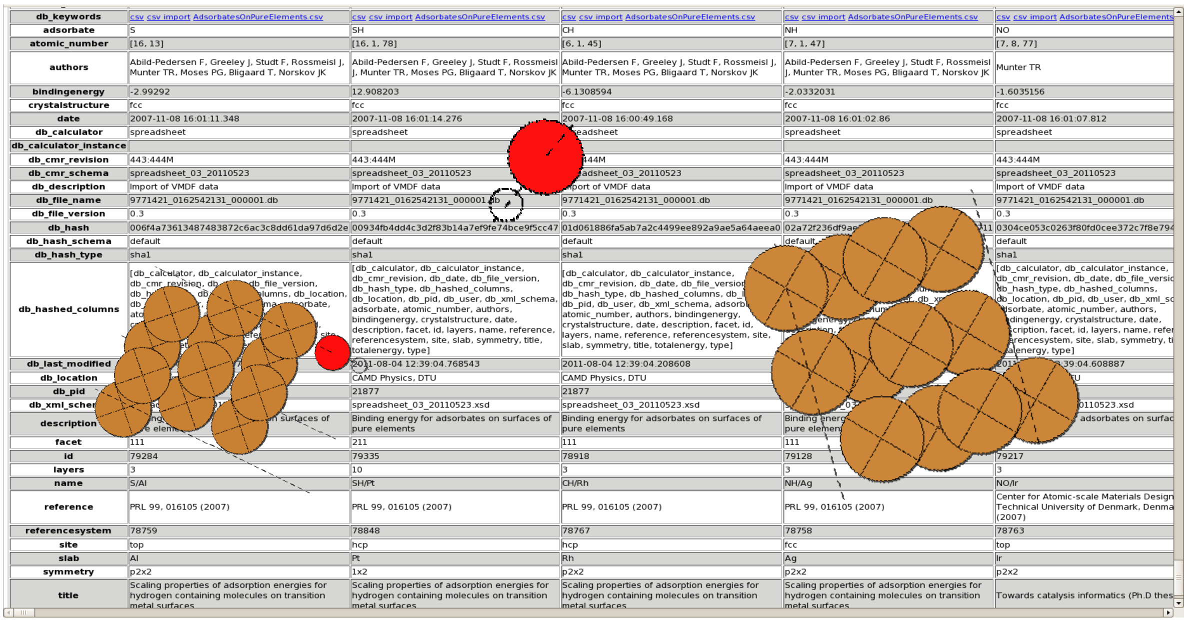
Publication date:
2012

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

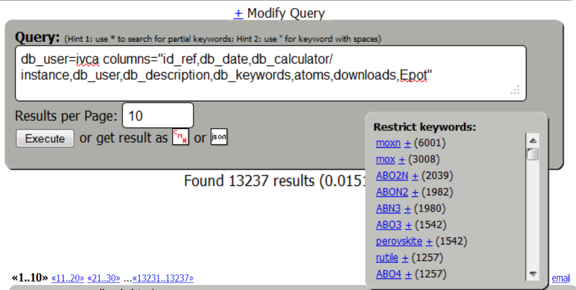
Citation (APA):
Landis, D. (2012). Computational Materials Repository. Technical University of Denmark, Center for Atomic-Scale Materials Physics.

Computational Materials Repository



db_keywords	csv csv import AdsorbatesOnPureElements.csv	csv csv import AdsorbatesOnPureElements.csv	csv csv import AdsorbatesOnPureElements.csv	csv csv import AdsorbatesOnPureElements.csv	csv csv import AdsorbatesOnPureElements
adsorbate	S	SH	CH	NH	NO
atomic_number	[16, 13]	[16, 1, 78]	[6, 1, 45]	[7, 1, 47]	[7, 8, 77]
authors	Abild-Pedersen F, Greeley J, Studt F, Rossmeisl J, Munter TR, Moses PG, Bligaard T, Norskov JK	Abild-Pedersen F, Greeley J, Studt F, Rossmeisl J, Munter TR, Moses PG, Bligaard T, Norskov JK	Abild-Pedersen F, Greeley J, Studt F, Rossmeisl J, Munter TR, Moses PG, Bligaard T, Norskov JK	Abild-Pedersen F, Greeley J, Studt F, Rossmeisl J, Munter TR, Moses PG, Bligaard T, Norskov JK	Munter TR
bindingenergy	-2.99292	12.908203	-6.1308594	-2.0332031	-1.6035156
crystalstructure	fcc	fcc	fcc	fcc	fcc
date	2007-11-08 16:01:11.348	2007-11-08 16:01:14.276	2007-11-08 16:00:49.168	2007-11-08 16:01:02.86	2007-11-08 16:01:07.812
db_calculator	spreadsheet	spreadsheet	[illegible]eadsheet	spreadsheet	spreadsheet
db_calculator_instance					
db_cmr_revision	443:444M	443:444M	[illegible]444M	443:444M	443:444M
db_cmr_schema	spreadsheet_03_20110523	spreadsheet_03_20110523	[illegible]adsheet_03_20110523	spreadsheet_03_20110523	spreadsheet_03_20110523
db_description	Import of VMDF data	Import of VMDF data	[illegible]port of VMDF data	Import of VMDF data	Import of VMDF data
db_file_name	9771421_0162542131_000001.db	9771421_0162542131_000001.db	9771421_0162542131_000001.db	9771421_0162542131_000001.db	9771421_0162542131_000001.db
db_file_version	0.3	0.3	0.3	0.3	0.3
db_hash	006f4a73613487483872c6ac3c8dd61da97d6d2e	00934fb4dd4c3d2f83b14a7ef9fe74bce9f5cc47	01d061886fa5ab7a2c4499ee892a9ae5a64aeea0	02a72f236df9ae[illegible]11	0304ce053c0263f80fd0cee372c7f8e794
db_hash_schema	default	default	default	default	default
db_hash_type	sha1	sha1	sha1	[illegible]	sha1
db_hashed_columns	[db_calculator, db_calculator_instance, db_cmr_revision, db[illegible] db_file_version, db_h[illegible] db[illegible]lumns, db_location, db[illegible]ma[illegible]orbate, ato[illegible]	[db_calculator, db_calculator_instance, db_cmr_revision, db_date, db_file_version, db_hash_type, db_hashed_columns, db_location, db_pid, db_user, db_xml_schema, adsorbate, atomic_number, authors, bindingenergy, crystalstructure, date, description, facet, id, layers, name, reference, referencesystem, site, slab, symmetry, title, totalenergy, type]	[db_calculator, db_calculator_instance, db_cmr_revision, db_date, db_file_versio[illegible] db_hash_type, db_hashed_columns, db_l[illegible] db_pid, db_user, db_xml_schema, adsorb[illegible] atomic_number, authors, bindingenergy, crystalstructure, date, description, facet, id, layers, name, reference, referencesystem, site, slab, symmetry, title, totalenergy, type]	[illegible]	[db_calculator, db_calculator_instance, db_cmr_revision, db_date, db_file_versio db_hash_type, db_hashed_columns, db_location, db_pid, db_user, db_xml_sc [illegible]dsorbate, atomic_number, authors, [illegible]dingenergy, crystalstructure, date, [illegible]cription, facet, id, layers, name, refer [illegible]erencesystem, site, slab, symmetry, ti [illegible]alenergy, type]
db_last_modified	[illegible]	2011-08-04 12:39:04.768543	2011-08-04 12:39:04.208608	[illegible]	201[illegible]:39:04.608887
db_location	[illegible]	CAMD Physics, DTU	CAMD Physics, DTU	[illegible]	[illegible]U
db_pid	[illegible]	21877	21877	[illegible]	[illegible]
db_xml_sche[illegible]	[illegible]	spreadsheet_03_20110523.xsd	spreadsheet_03_20110523.xsd	[illegible]	[illegible]0110523.xsd
description	[illegible] surfaces of [illegible]e elem[illegible]	Binding energy for adsorbates on surfaces of pure elements	Binding energy for adsorbates on surfaces of pure elements	Binding energ[illegible] pure elemen[illegible]	[illegible] adsorbates on surfac
facet	111	211	111	111	[illegible]
id	79284	79335	78918	79128	79217
layers	3	10	3	3	3
name	S/Al	SH/Pt	CH/Rh	NH/Ag	NO/Ir
reference	PRL 99, 016105 (2007)	PRL 99, 016105 (2007)	PRL 99, 016105 (2007)	PRL 99, 016105 (2007)	Center for Atomic-scale Materials Design Technical University of Denmark, Denma (2007)
referencesystem	78759	78848	78767	78758	78763
site	top	hcp	hcp	fcc	top
slab	Al	Pt	Rh	Ag	Ir
symmetry	p2x2	1x2	p2x2	p2x2	p2x2
title	Scaling properties of adsorption energies for hydrogen containing molecules on transition metal surfaces	Scaling properties of adsorption energies for hydrogen containing molecules on transition metal surfaces	Scaling properties of adsorption energies for hydrogen containing molecules on transition metal surfaces	Scaling properties of adsorption energies for hydrogen containing molecules on transition metal surfaces	Towards catalysis informatics (Ph.D thes

David Dominic Landis
Ph.D. Thesis
Januar 2012



+ Modify Query

Query: (Hint 1: use * to search for partial keywords; Hint 2: use " for keyword with spaces)

```
db_user=ivca columns="id_ref,db_date,db_calculator/
instance,db_user,db_description,db_keywords,atoms,downloads,Epot"
```

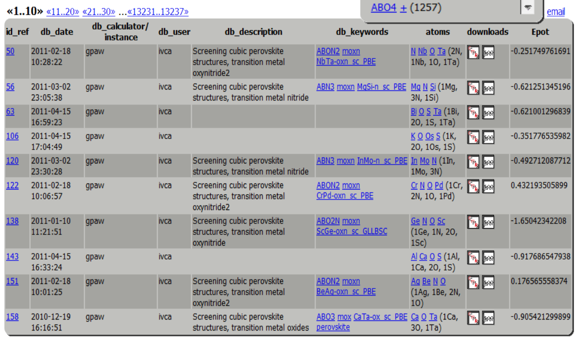
Results per Page: 10

Execute or get result as or json

Found 13237 results (0.015

Restrict keywords:

moxn +	(6001)
mox +	(3008)
ABO2N +	(2039)
ABON2 +	(1982)
ABN3 +	(1980)
ABO3 +	(1542)
perovskite +	(1542)
rutile +	(1257)
ABO4 +	(1257)



«1..10» «11..20» «21..30» ...«13231..13237» email

id_ref	db_date	db_calculator/ instance	db_user	db_description	db_keywords	atoms	downloads	Epot
50	2011-02-18 10:28:22	gpaw	ivca	Screening cubic perovskite structures, transition metal oxynitride2	ABON2 moxn NbTa-oxn_sc_PBE	N Nb O Ta (2N, 1Nb, 1O, 1Ta)		-0.251749761691
56	2011-03-02 23:05:38	gpaw	ivca	Screening cubic perovskite structures, transition metal nitride	ABN3 moxn MgSi-n_sc_PBE	Mg N Si (1Mg, 3N, 1Si)		-0.621251345196
63	2011-04-15 16:59:23	gpaw	ivca			Bi O S Ta (1Bi, 2O, 1S, 1Ta)		-0.621001296839
106	2011-04-15 17:04:49	gpaw	ivca			K O Os S (1K, 2O, 1Os, 1S)		-0.351776535982
120	2011-03-02 23:30:28	gpaw	ivca	Screening cubic perovskite structures, transition metal nitride	ABN3 moxn InMo-n_sc_PBE	In Mo N (1In, 1Mo, 3N)		-0.492712087712
122	2011-02-18 10:06:57	gpaw	ivca	Screening cubic perovskite structures, transition metal oxynitride2	ABON2 moxn CrPd-oxn_sc_PBE	Cr N O Pd (1Cr, 2N, 1O, 1Pd)		0.432193505899
138	2011-01-10 11:21:51	gpaw	ivca	Screening cubic perovskite structures, transition metal oxynitride	ABO2N moxn ScGe-oxn_sc_GLLBSC	Ge N O Sc (1Ge, 1N, 2O, 1Sc)		-1.65042342208
143	2011-04-15 16:33:24	gpaw	ivca			Al Ca O S (1Al, 1Ca, 2O, 1S)		-0.917686547938
151	2011-02-18 10:01:25	gpaw	ivca	Screening cubic perovskite structures, transition metal oxynitride2	ABON2 moxn BeAg-oxn_sc_PBE	Ag Be N O (1Ag, 1Be, 2N, 1O)		0.176565558374
158	2010-12-19 16:16:51	gpaw	ivca	Screening cubic perovskite structures, transition metal oxides	ABO3 mox CaTa-ox_sc_PBE perovskite	Ca O Ta (1Ca, 3O, 1Ta)		-0.905421299899

Computational Materials Repository

David Dominic Landis

Kongens Lyngby 2012

Technical University of Denmark
Center for Atomic-scale Materials Design
Building 311, DK-2800 Kongens-Lyngby, Denmark
www.camd.dtu.dk

Abstract

The ongoing growth in computing power enables researchers to perform such a large number of simulations that cannot be analyzed with paper and pencil any more. Simple approaches of processing data: ordering the calculations in directories and using a script to create a spreadsheet or a small database have to be redesigned for every new project. Sharing intermediate data with collaborators can be cumbersome and when publishing on the Internet specially tailored infrastructure has to be set up.

Due to the diverse and changing landscape of electronic structure codes and methods there is no unique way of storing, collecting and presenting results. However there are many partial solutions: VMDF (paper D) a tool to filter and analyze aggregated sets of electronic structure data presents a first step towards user-friendly analysis of data. The Inorganic Crystal Structure Database ICSD[1, 2], collects very specific data and makes it accessible through a web interface; AflowLib (Ab-initio Electronic Structure Library) [3] provides access to structure properties of many compounds on the Internet.What is missing is a system that is Open Source Software, generic enough to support different codes, different abstraction levels and enables users to analyze their own results, and allows to share data with collaborators.

The approach of the Computational Materials Repository (CMR) is to convert data to an internal format that maintains the original variable names without insisting on any semantics. Imported data can be implicitly grouped by user criteria and therefore maintain their natural connection in the database as well. Automatic data analysis is enabled through agents that analyze and group data based on predefined rules. Small projects can be handled without the need of database software while bigger projects one can use to improve performance.

CMR enables one to create templates for the collection and analysis of data independently of the electronic structure code, simplifies screenings involving a lot of calculations, allows one to perform automatic analysis of data based on taxonomy, tags and keywords, provides the ability to share data with collaborators and maintains the link from the derived to the original data.

Resumé

Den igangværende vækst i computerkraft gør det muligt for forskere at udføre et såstort antal simuleringer, at det ikke længere er muligt at analysere med papir og blyant. Enkle tilgange til behandling af data: samling af beregninger i mapper og brug af et script til at generere et regneark eller en lille database måredesignes for hvert nyt projekt. Deling af intermediær data med samarbejdspartnere kan være besværligt og ved publikation påinternettet skal specifikt skræddersyede infrastrukturer opsættes.

Grundet det mangeartede og foranderlige landskab af koder og metoder til elektronstruktur-beregninger findes ingen unik måde at gemme, samle og præsentere resultater på. Der findes imidlertid mange delvise løsninger: VMDF (paper D) er et værktøj til filtrering og analyse af aggregerede sæt af elektronstruktur data og repræsenterer et første skridt påvejen mod brugervenlig analyse af data. The Inorganic Crystal Structure Database ICSD[1, 2], samler meget specifikke data og gør dem tilgængelige via et web-interface. AflowLib (Ab-initio Electronic Structure Library) [3] giver adgang til struktur-egenskaber for mange kemiske forbindelser påinternettet.

Det som mangler er et system som er Open Source Software, er generelt nok til at understøtte forskellige koder og forskellige abstraktions-niveauer, og som gør det muligt for brugere at analysere deres egne resultater og tillader deling af data med samarbejdspartnere.

Fremgangsmåden i Computational Materials Repository (CMR) er at konvertere data til et internt format, som bibeholder de originale variable uden at insistere pånogen semantik. Importerede data kan implicit grupperes efter brugerbestemte kriterier, og derved ogsåbibeholde deres naturlige forbindelse

i databasen. Automatisk dataanalyse muliggøres af agenter som analyserer og grupperer data baseret påprædefinerede regler. Småprojekter kan håndteres uden brug af database-software, mens større projekter kan bruges til forøget ydeevne.

CMR gør det muligt at oprette skabeloner til samling og analyse af data uafhængigt af den givne elektronstruktur-kode, forenkler screening af store mængder beregninger, muliggør automatisk analyse af data baseret påtaksonomi, tags og keywords, giver mulighed for at dele data med samarbejdspartnere og bibeholder forbindelsen fra det afledte til det originale data.

Preface

This thesis is submitted in candidacy for the Ph.D. degree from the Technical University of Denmark (DTU). The work presented was carried out at the Center for Atomic-scale Materials Design (CAMd) from June 2008 to August 2011 under the supervision of Professor Jens K. Nørskov with co-supervisor Professor Thomas Bligaard and after Professor Jens K. Nørskov left with Professor Karsten W. Jacobsen as supervisor.
NABIIT and CAMd which is funded by The Lundbeck Foundation are gratefully acknowledged for their financial support.

I would like to thank my supervisor Karsten Wedel Jacobsen for his excellent supervision and guidance throughout my Ph.D., Thomas Bligaard and Jens K. Nørskov for sharing their vision with me and Thomas for telling me about this really interesting project and for the many discussions at the beginning and at the end of my Ph.D.
I would like to thank Jens Strabo Hummelshøj for all the lively discussion about how things should be done.
I would like to thank Marcin Dułak for all his feedback and his interest in the project.
I would like to thank Jefferey Greeley, Svetlozar Nestorov and Glen Ferguson for all the good discussions.
I would like to thank all people that commented on my project and submitted bugs and suggestions.

I would like to thank Tao Jiang for whom I'm extraordinarily happy to have shared office, flat and many great experiences with.
I would like to thank all my office mates Jess Wellendorff, André Kelkkanen,

Federico Calle Vallejo, Keld Lundgård, Christen Kjær and all the others who at some point inhabited office 254 for the good climate.
I would like to thank Jess Wellendorff for translating the abstract.
I would also like to thank Marcin Dułak for proof-reading my complete thesis and as well Tao Jiang and Steen Lysgård.
I would like to thank Juan Shen, Isabela Man, George Tritsaris, Jun-Yan, Ivano Castelli, Vladimir Tripkovic, Ask Hjorth Larsen, Mohammedreza Karamad, Samira Siahrostami, Paweł Zawadzki, Marco Vanin, Elisa Fiordaliso and Tobias Johansson, Falco Hüser, Duncan Mowbray, Anca Paduraru and all the people at CAMd and CINF for the great atmosphere.
A special thanks to the administrative staff Marianne Ærsøe, Stavroula Nielsen, Henning Bo Nicolajsen, Ole Holm Nielsen and Helle Wedel Wellejus which do and did an awesome job keeping CAMd running.
Last but not least I would like to thank Kate for her loving support.

Lyngby, January 2012

David Dominic Landis

Papers included in the thesis

A **The Computational Materials Repository**
D. D. Landis, J. S. Hummelshøj, S. Nestorov, J. Greeley, M. Dułak, T. Bligaard, J. K. Nørskov and K. W. Jacobsen
Accepted by Comput. Sci. Eng., 2012

B **Computational Screening of Perovskite Metal Oxides for Optimal Solar Light Capture**
I. E. Castelli, T. Olsen, S. Datta, D. D. Landis, K. S. Thygesen, S. Dahl, K. W. Jacobsen
Energy Environ. Sci., vol. 5, 2012

C **Density functional theory based screening of ternary alkali-transition metal borohydrides: A computational material design project**
J. S. Hummelshøj, D. D. Landis, et al.
J. Chem. Phys., vol. 131, 2009

D **Virtual Materials Design using Databases of Calculated Materials Properties**
T. R. Munter, D. D. Landis,F. Abild Pedersen, G. Jones, S. Wang and T. Bligaard
Comput. Sci. Disc., vol.2, 2009

Acronyms, Terms and Definitions

term	**short explanation**	**chapter**
add/commit/ submit/upload	the process of adding db-files to the db-file repository; from the db-file repository the data is uploaded to the MySQL database	3.3.4.a
agent	autonomous process for data analysis or tasks in the background	2.2, 2.9.3, 3.3.3.a
calculation	a short name for an *electronic structure calculation* performed by a code like GPAW or VASP	
code	*electronic structure simulator* e.g. GPAW or VASP	
CLI	the command line interface allows to run common tasks like showing content of db-files to be executed in a linux shell	2.5,3.3.1.a
CMR database	*see* database	
CMR schema	*see* schema	
cluster	a computer cluster or super computer that executes jobs (scripts)	
database	with database the MySQL database with the CMR database schema is meant	3.3.2.c
commit	*see* add	

db-file (cmr-files)	db-file is the internal file format of CMR; in the future these files will be called cmr-files to avoid naming conflicts with other file formats	3.3.3.b
db-file repository	the db-file repository stores db-files that are later uploaded to the MySQL database	3.3.2.a
field(s)	a field is a name/value pair e.g. energy/-0.1	2.1
group	a group contains a collection of results (calculations); it is also possible that groups contain other groups	2.2, 2.7
keyword(s)	keywords are used to identify calculations and can be added to all the data uploaded to the database	2.1
mapping	the process of renaming a field or changing the unit of the value dynamically when reading a db-file	2.9.2
MySQL database	*see* CMR database	
PHP/HTML (user) interface	a user interface of CMR that runs on a web server and is accessed with a web browser	2.3, 3.3.1.c
PHPUI	*see* PHP/HTML (user) interface	
PUI	*see* python (user) interface	
python (user) interface	a user interface of CMR	3.3.1.b
schema	the database/ CMR schema	3.3.3.c, 3.3.2.c
submit	*see* add	
SiGUI	a plug-in for Silo	3.3.1.d.1
Silo	a user interface that connects to the CMR database	3.3.1.d
unique identifier	a unique identifier is created for every db-file; this identifier is used to identify group members	2.6
UI	*see* unique identifier	
upload	*see* add	

Contents

Chapter 1

Introduction

The design of novel and versatile materials is of great importance for our society. This is is reflected by the strong focus on discovering new materials for energy conversion and storage to provide a sustainable alternative to the fossil-based fuel economy. Atomic-scale calculations are becoming increasingly important in strengthening our ability towards meeting this challenge, as it over time has provided an ever-improving alternative to expensive experiments. As computational resources become more readily available and the methods more efficient, studies can be performed on complex systems. This poses a challenge in terms of systematic storage, retrieval and analysis of the results. If there is no automated process or predefined schema for the collection and analysis, every researcher will spend more and more time on administrating his/her data instead of deriving useful information. Other time consuming tasks that have to be done repeatedly are sharing intermediate results with collaborators, archiving data and documentation how it was gathered and making it available on the Internet.

Institutes and research facilities have a particular interest in aggregating and storing data in a uniform way: if done right they not only have access to former research data, but can optimize future calculations by finding better initial guesses of atomic positions, data mine older data or continue a project seamlessly.

In biological research, scientists realized early that sharing of the data, through commonly available databases improves research and leads to faster progress in the field. The National Center for Biotechnical Information (NCBI)[4] has the biggest and the most cited database in the molecular biology and medical field - and its free. The database was initiated in 1988 by the American government and consists of subsets of smaller databases that hold information about gene and protein structure, genetic variants, common domains, et cetera. Additionally NCBI links all published articles to relating gene/protein datasets. By typing the name of a gene in the search box, NCBI searches all its databases and provides cross-linked results to relating data sets which enables further investigations and analyses. Researchers are also able to upload their own data. Nowadays NCBI is an integral part in daily routine of biologist researchers.

Other biological databases are Ensemble[5], which stores information about the genome, or the Protein Data Bank (PDB)[6] - which contain published 3D structures of proteins; a good example of collaborate efforts is the Human Genome Project[7] that was initiated in 1990 and aimed at sequencing the whole human genome. Many institutes around the world were participating in sequencing, analyzing, annotating and submitting data.

The electronic structure calculation community is not there yet. Especially we are far from a cross-linked freely available database where researchers can upload their data and put it in context with already available data in order to derive new results - however partial solutions exist: The Virtual Materials Design Framework (VMDF) (paper D), a tool to filter and analyze aggregated sets of electronic structure data introduces a first step towards user-friendly analysis of data, the CAMd database[8] or AFlowLib[3] that present a selection of aggregated data on the Internet and make it available through a web interface - or ICSD, the Inorganic Crystal Structure Database[1, 2] that collects validated experimentally determined data, and provides a software tool and a web interface to perform searches - unfortunately not for free. During the last eight months more elaborate tools and databases have seen the light of day: ESTEST[9] aims at validation and verification of electronic structure calculations of different codes; Materials Genome[10, 11] has a high-throughput infrastructure to perform screenings and presents results through toolboxes that are accessible on the Internet - and the Quixote[12, 13] project that focuses on quantum chemistry data presents a collaborative open source framework to collect and process data from different sources.

The reasons why we are not there is that it is hard to find a common applicable scheme that is able to cover most aspects. The electronic structure data is for instance more heterogeneous - there are numerous file formats - and the the analysis depends greatly on the perused goal. The challenges of handling data in an efficient and reusable way can be divided in a few mature tasks as represented in Fig. 1.1.

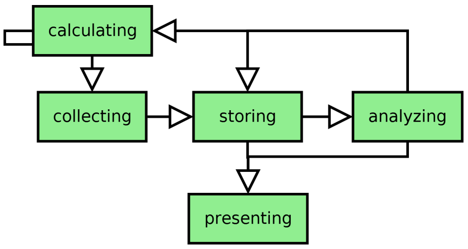
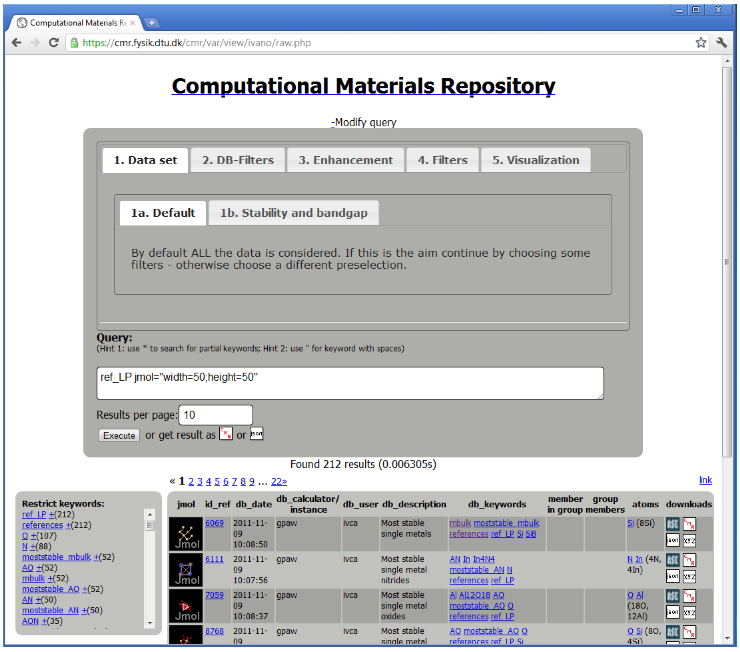


Figure 1.1: Major tasks when aggregating data - the arrows represent the work flow. **Calculating** produces results that need to be **collected** in a commonly readable format. These results could then be **stored** in a database for further **analysis**. The **presentation** can be done in a web-interface or any other tool that is able to retrieve data from the storage. Important is to notice that analysis can produce derived data that should be added to the storage as well.

With the Computational Materials Repository (CMR) that is presented in this thesis we provide a modular open source system that addresses the challenges of collecting, storing, making data available through interfaces and propose and implement basic ideas how to perform analysis. The focus is in particular on DFT codes because they represent a favorable trade-off between speed and accuracy for the treatment of “few-hundred-atom” systems highly relevant for understanding physical and chemical properties of materials. We propose a file-format (db-files) that holds extracted data syntactically and semantically as close as possible to the original output file - and introduce groups to indicate connections between data. The Python user interface allows to search and query data which then can be analysed. In order to cope with different assumptions on names and units, CMR implements mappings. A mapping enables to rename and convert for example the field `CartesianPositions` in Bohr units, to `positions` in Angstrom.

To get a user-friendly way of looking at the data, the PHP/HTML interface presents the data with the help of a MySQL database and a webserver as shown in Fig. 1.2. When the data is published it is possible to create a custom view as shown in Fig. 1.3 to make the results easier accessible for public.


Computational Materials Re

https://cmr.fysik.dtu.dk/cmr/var/view/ivano/raw.php

Computational Materials Repository

-Modify query

1. Data set | 2. DB-Filters | 3. Enhancement | 4. Filters | 5. Visualization

1a. Default | 1b. Stability and bandgap

By default ALL the data is considered. If this is the aim continue by choosing some filters - otherwise choose a different preselection.

Query:
(Hint 1: use * to search for partial keywords; Hint 2: use " for keyword with spaces)

ref_LP jmol="width=50;height=50"

Results per page: 10

Execute or get result as CMR or json

Found 212 results (0.006305s)

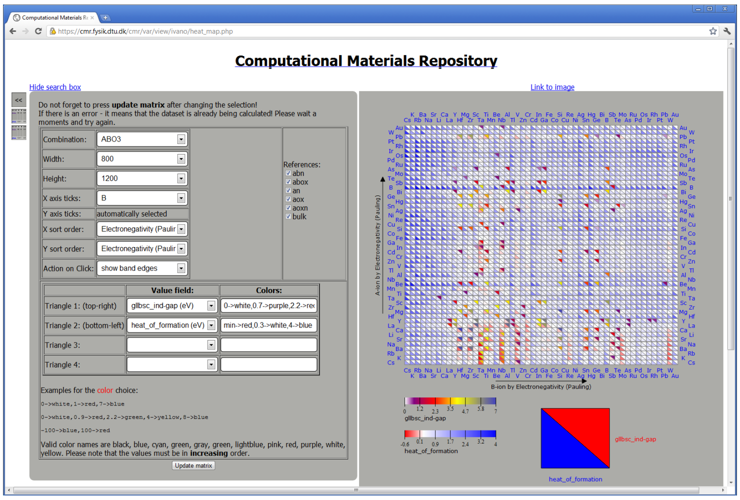
« **1** 2 3 4 5 6 7 8 9 ... 22»

link

Restrict keywords:
ref_LP +(212)
references +(212)
O +(107)
N +(88)
moststable_mbulk +(52)
AO +(52)
mbulk +(52)
moststable_AO +(52)
AN +(50)
moststable_AN +(50)
AON +(35)

jmol	id_ref	db_date	db_calculator/ instance	db_user	db_description	db_keywords	member in group	group members	atoms	downloads
Jmol	6069	2011-11-09 10:08:50	gpaw	ivca	Most stable single metals	mbulk moststable_mbulk references ref_LP Si Si8			Si (8Si)	ASE CMR json xyz
Jmol	6111	2011-11-09 10:07:56	gpaw	ivca	Most stable single metal nitrides	AN In In4N4 moststable_AN N references ref_LP			N In (4N, 4In)	ASE CMR json xyz
Jmol	7059	2011-11-09 10:08:37	gpaw	ivca	Most stable single metal oxides	Al Al12O18 AO moststable_AO O references ref_LP			O Al (18O, 12Al)	ASE CMR json xyz
	8768	2011-11-09	gpaw	ivca	Most stable single metal	AO moststable_AO O references ref_LP Si			O Si (8O, 4Si)	ASE CMR

Figure 1.2: The PHP/HTML interface finds data based on keywords and fields, suggests related keywords and provides ways to download the data.


Computational Materials Re

https://cmr.fysik.dtu.dk/cmr/var/view/ivano/heat_map.php

Computational Materials Repository

Hide search box

Link to image

Do not forget to press **update matrix** after changing the selection!
If there is an error - it means that the dataset is already being calculated! Please wait a moments and try again.

Combination:	ABO3
Width:	800
Height:	1200
X axis ticks:	B
Y axis ticks:	automatically selected
X sort order:	Electronegativity (Paulir
Y sort order:	Electronegativity (Paulir
Action on Click:	show band edges

References:
☑ abn
☑ abox
☑ an
☑ aox
☑ aoxn
☑ bulk

	Value field:	Colors:
Triangle 1: (top-right)	gllbsc_ind-gap (eV)	0->white,0.7->purple,2.2->re
Triangle 2: (bottom-left)	heat_of_formation (eV)	min->red,0.3->white,4->blue
Triangle 3:		
Triangle 4:		

Examples for the color choice:

```
0->white,1->red,7->blue
0->white,0.9->red,2.2->green,4->yellow,8->blue
-100->blue,100->red
```

Valid color names are black, blue, cyan, green, gray, green, lightblue, pink, red, purple, white, yellow. Please note that the values must be in **increasing** order.

Update matrix

A-ion by Electronegativity (Pauling)
B-ion by Electronegativity (Pauling)
gllbsc_ind-gap
heat_of_formation
gllbsc_ind-gap
heat_of_formation

Figure 1.3: A customized view of the PHP/HTML interface to present published data in a heat-map.

The focus for the development of CMR was lied on generality, modularity and extensibility. This leaves a lot of room for performance tuning like using a compiled language to verify the db-files or only uploading data to the database that is actually needed.

The Computational Materials Repository is developed in collaboration between J. Greeley from the Argonne National Laboratory and S. Nestorov from the University of Chicago and J. S. Hummelshøj, T. Bligaard, J. K. Nørskov from SUNCAT Center for Interface Science and Catalysis in Standford and us - Center for Atomic-scale Materials Design. CMR is part of Quantum Materials Informatics Project[14], which aims at establishing the core technology for integrated computational materials design.

This thesis begins with an introduction by example to present the capabilities and usage of CMR followed by a chapter that provides details about the challenges, the work flows and technical information. The chapter with the case studies then discusses projects where CMR was used.

Chapter 2

Introduction and usage of CMR

This chapter shows what the Computational Materials Repository (CMR) can do for you with concrete examples and explanations of the user interfaces. The subsequent chapter focuses on concepts and the inner workings.

We shall first briefly give an overview of all the functionalities of CMR and then move on to more concrete illustrations of its usage.

All the different components of CMR are depicted in Fig. 2.1. CMR simplifies the handling of heterogeneous data from electronic structure codes. It provides the same interface for handling the data independent of the original file format (and file names). The general interface makes scripting easy and reusable. Users of electronic structure codes are typically interested in obtaining and saving quantities like atomic positions, energies, and forces. Often though also code specific input parameters and results are relevant. For this reason CMR extracts most of the variables from the original output file and makes use of the ASE[15] (Atomic Simulation Environment) interface (if applicable) to get the relevant data in a unified representation in terms of variable names and units.

The extracted data is stored in a file format that we call **db-file**. db-files can be read on any linux machine that has Python[16] and CMR installed. The

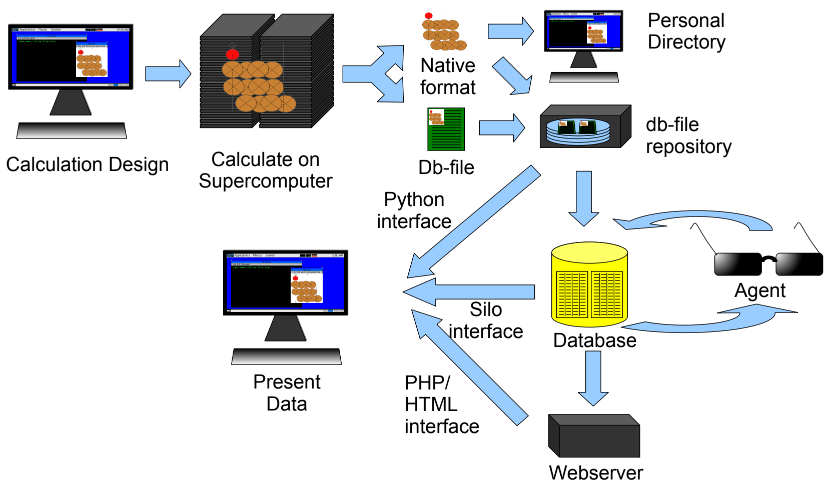


Figure 2.1: Overview over the infrastructure and the workflow of the data in CMR.

command line interface enables simple everyday tasks on these files like viewing content and performing basic editing. Normally all db-files are moved to a single directory that we call the **db-file repository**(3.3.2.a). With the CMR **python interface** queries can be run on the database as shown in section 2.2.

The data cannot only be stored in a collection of files. As depicted in figure 2.1 the data from the db-files can be uploaded to a database and queried with the python interface which will result in faster searches, because the data is indexed. More elaborate installations can make use of the **PHP/HTML interface** that provides a graphical user interface for searching and viewing data while **Silo**[1] features a workbench to analyze data. Last but not least so-called agents may run in the background – either invisible from the users or under user control – and perform certain tasks as for example grouping of db-files or preparing data needed by the user interfaces.

Currently CMR supports the import of GPAW, Dacapo, VASP and ASE trajectory files. Additionally CSV[17] files are supported which can be read and written by OpenOffice or Microsoft Excel.

[1]**Silo** was initiated and is maintained by Jens Strabo Hummelshøj, at the time of writing a post-doc at SUNCAT Center for Interface Science and Catalysis at Stanford

2.1 Working with CMR

Working with a database is quite different from the common approach of storing data in a file system: files and directories are non-existent and therefore the data has to be identified in an other way. CMR implements an abstraction layer that simplifies finding of results by **keywords** or **fields**. A field is simply a name/value pair as for example energy/12.0 or program/gpaw.

In this section we discuss how to organize the data to be able to do the same as with the "old" approach and additionally profit from storing it with CMR in a database. Please note that the **python interface** supports to querying the **db-file repository** as well as the **database** (MySQL database) while **agents** work only with the database. However this does not change the way the data is organized.

There are few requirements that need to be identified in order to work efficiently with the database: First, the results need to be found after putting them into the database. Keywords replace directory and file names. For instance the file with the path 211surfaces/Ag211/edge/H.gpw would get the keywords *211surface*,*Ag211*,*edge*. There is no need to put *H* as a keyword, for finding purposes, because we can search/restrict by atom type. However, if the count of the atoms matters as for N2 then it would make sense to include it. (The reason is that when looking for N2 we would also get the results with the single atom N.) The advantage of keywords over directory names is that the order doesn't matter and they enable to look at arbitrary subsets. When the two keywords *211surface* and *edge* and the requirement of presence of the atom H are combined in a query, we find all kinds of 211 surfaces with an H atom at the edge position. To look for these files in subdirectories would be considerably more cumbersome. Second, we would like to read the data in a similar way as the native output file. This is achieved with the python user interface as described later in section 2.4 or with the interface that views the data in a web browser as shown already in the introduction in Fig. 1.2. Third we profit from the database capability to search efficiently for the defined keywords and fields. Fourth, the results should provide more information than just the numbers from the output file. db-files are capable of storing scripts, in-/or output files, calculation parameters and custom fields as for example surface=211. Fifth, the results should be traceable. By creating groups (described later in the section 2.7) that reference the used data is conserved.

One might wonder what is the difference between the keyword *211surface* and the field surface=211, because both define the same data set. The different characteristics can be seen when grouping calculations according to criteria specific criteria. The chemisorption energy E_{chem} is calculated as $E_{chem} = E_Z - E_X - E_Y$, where E_X the adsorbate X in the gas-phase, E_Y the clean surface with atoms Y and E_Z is the total energy of surface with atoms Y with the adsorbate Z. If we had only the keywords, then we would have to know every single keyword and loop manually over all possible combinations. In pseudo code it would look as follows:

```
for surface in [111,211,...]:
    for adsorbate in [H, O, ...]:
        find result with keyword surface+adsorbate
         E_chem = ...
```

This is not efficient because every possible combination has to be checked - even if there is no data in the database and because the actually available surfaces and adsorbates have to be known. They cannot automatically be determined from the database.

A better way is to use the keyword to identify a certain type of calculation and the fields to combine them. This can be written as the following list of rules:

- X.keywords *contains* adsorbate
- Y.keywords *contains* surface
- Z.keywords *contains* surface+adsorbate
- X.ads=Z.ads
- Y.surf=Z.surf

These rules can then be translated to MySQL, the query language for the database. This approach is more general since the surfaces and adsorbates are determined automatically; the only thing that needs to be specified is the name of the fields (*ads* for the adsorbate and *surf*, for the surface). CMR provides a tool called GroupingAgent that translates the list of rules into query language. Its usage is explained in the next section 2.2.

To summarize: keywords *identify* a calculations or a certain type of calculation, while fields enable efficient *combination/grouping*. Groups show a *connection* between individual calculations or explain what calculations were involved to derive a result. Generally one would add a keyword to identify the type of calculation and fields for the properties that are needed to make the groups or to store additional information about the calculation.

2.2 Step by Step

In this section more concrete examples of how to work with CMR are presented. In order to actually run the presented python code, the user needs to have CMR installed – or have access to an institute that has deployed it. The CMR wiki and the installation instructions are available from `https://wiki.fysik.dtu.dk/cmr`.

The example takes its offset in a data set acquired as a computational screening for materials which can use solar light to split water - one potential way of producing hydrogen fuel. How the calculations are performed is described in detail in Paper B "Computational Screening of Perovskite Metal Oxides for Optimal Solar Light Capture". In the context here the following background should be sufficient.

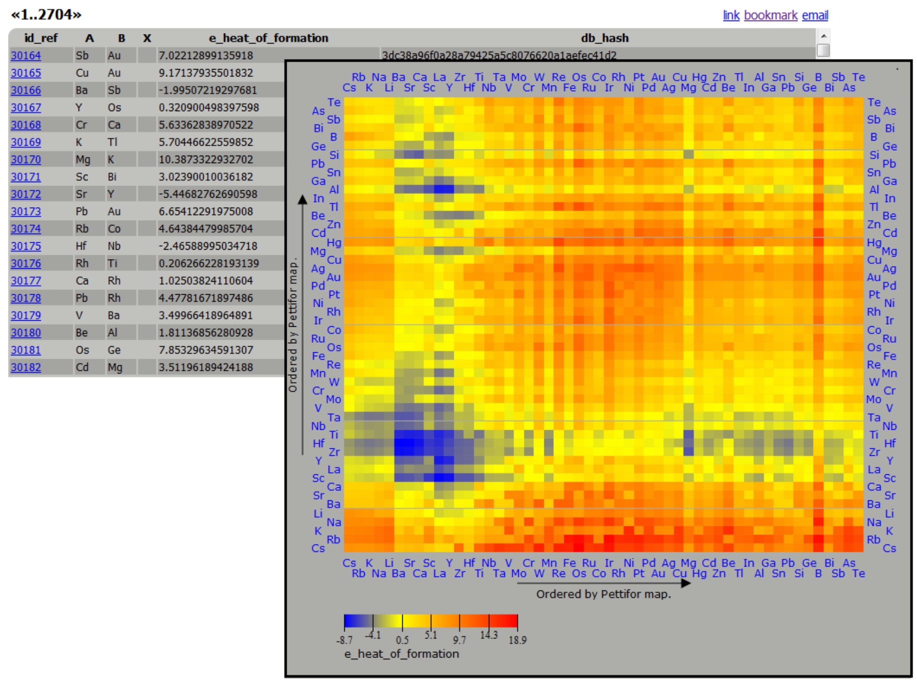
For a material to be able to split water based on solar light, a number of conditions have to be fulfilled. First of all the material of course has to be stable also when surrounded by water. Furthermore, the electronic energy gap has to be in the right range (1.5-3 eV) so that the generated electron-hole pairs have sufficient energy to perform the water splitting (for this a large gap is preferred), but at the same time the gap should be small to increase the amount of light absorbed. There are some further conditions related to the positions of the band edges relative to the redox potentials for water splitting but we do not need to go into that here. The class of materials which have been investigated are mainly oxides (and oxynitrides) in the cubic perovskite structure. The interested reader is referred to more details in Paper B at the end of this thesis.

Here we focus on oxides in the perovskite structure which has the composition ABO_3 with A and B being metals. The above mentioned stability is investigated by defining the *heat of formation* (hof) E_{hof} of the ABO_3 compound as

$$E_{hof} = E_{ABO_3} - E_A - E_B - 3E_O$$

and we will show how this can be retrieved from the database as a list and also export it as a csv-file[17] (Comma Separated Values). To get an overview, the results should be displayed in a heat map with the PHP/HTML interface as shown in figure 2.2.

However, to begin with the beginning we should first get some data into CMR. The perovskite calculations have all been done with GPAW and this program is



«1..2704»

link bookmark email

id_ref	A	B	X	e_heat_of_formation	db_hash
30164	Sb	Au		7.02212899135918	3dc38a96f0a28a79425a5c8076620a1aefec41d2
30165	Cu	Au		9.17137935501832	
30166	Ba	Sb		-1.99507219297681	
30167	Y	Os		0.320900498397598	
30168	Cr	Ca		5.63362838970522	
30169	K	Tl		5.70446622559852	
30170	Mg	K		10.3873322932702	
30171	Sc	Bi		3.02390010036182	
30172	Sr	Y		-5.44682762690598	
30173	Pb	Au		6.65412291975008	
30174	Rb	Co		4.64384479985704	
30175	Hf	Nb		-2.46588995034718	
30176	Rh	Ti		0.206266228193139	
30177	Ca	Rh		1.02503824110604	
30178	Pb	Rh		4.47781671897486	
30179	V	Ba		3.49966418964891	
30180	Be	Al		1.81136856280928	
30181	Os	Ge		7.85329634591307	
30182	Cd	Mg		3.51196189424188	

Cs Rb K Na Li Ba Sr Ca Sc La Y Zr Hf Ti Nb Ta V Mo Cr W Mn Re Fe Os Ru Co Ir Rh Ni Pt Pd Au Ag Cu Mg Hg Cd Zn Be Tl In Al Ga Sn Pb Si Ge B Bi Sb As Te
Ordered by Pettifor map.
Ordered by Pettifor map.
-8.7
-4.1
0.5
5.1
9.7
14.3
18.9
e_heat_of_formation

Figure 2.2: A table showing the numeric results and a heatmaps with the heat of formation for ABO3. The elements are ordered along the axis according to Pettifor's stringing of the Periodic Table.

particularly well integrated with CMR so data transfer is easy. GPAW allows the use of the native `GPAW.write` function to directly produce a db-file as shown in figure 2.3. This has the advantage that the result is read directly from memory. Furthermore, at CAMD the db-file repository is a directory from where the data is automatically uploaded to the database, but CMR can also be used directly with the collection of db-files.

As is fairly obvious from the file, the format allows for the definition of a number of keywords ("ABO3" and "CsZnO3") as well as script-files and possibly other files that should be saved with this entry. It is also possible to define new so-called *fields* which are name–value pairs. The last field in the example for example has the name "X" with the value "O3". Some of the information carried by the fields or keywords are in this example somehow redundant. It is of course possible just from the identity of the atoms and their positions to recover which metal atom is in fact the "A" atom in the perovskite structure, but it may be convenient sometimes to add the information explicitly for fast and easy retrieval. If other

```
import cmr ...
calc = GPAW() ...
cmr_params = {
  "db_keywords":["ABO3", "CsZnO3"],
  "db_scripts":["CsZnO3.py"],
  "db_files":["CsZnO3.txt"],
  "name":"CsZnO3",
  "A":"Cs",
  "B":"Zn",
  "X":"O3"
}
calc.write(".db", cmr_params=cmr_params)
```

Figure 2.3: A small python script to add an arbitrary result to the db-file repository. GPAW provides the ability to use its native write function.

programs than GPAW are used or if one wants to “manually” add information to the db-files this can be done as shown in figure 2.4.

```
import cmr

cmr_params = {
    "input":"CsZnO3.xxx",
    "output":".db",
    "db_keywords":["ABO3", "CsZnO3", "hof"],
    "db_scripts":["CsZnO3.py"],
    "db_files":["CsZnO3.txt"],
    "name":"CsZnKO3",
    "A":"Cs",
    "B":"Zn",
    "X":"O3"
}
cmr.convert(cmr_params)
```

Figure 2.4: A small python script to add an arbitrary result to the db-file repository.

All extra fields and keywords are defined in a python dictionary and passed as an extra argument to `cmr.convert`. "input" specifies the file that should be imported and "output" the name of the output file - using ".db" as a name signals that we are not interested in a real filename and would like the output to be written directly to the db-file repository (from where it will be added to the database). "db_keywords" is a list of keywords that identify this calculation, db_scripts and db_files take a list of filenames that will be included - and finally "A", "B" and "X" are user defined fields.

The calculation of the heats of formation require input from several different

GPAW calculations. The link between different calculations in CMR is established by creating so-called *groups* of calculations: The resulting calculated energy, $E_{hof,CsZnO_3}$, will be linked to a group containing the calculations for $CsZnO_3$, Cs, Zn, and O. When loading the data into CMR they have been equipped with keywords and fields as illustrated in figure 2.3, so the A- and B-metal atoms in a given perovskite calculation can be referred to using the fields "A" and "B". The keywords make finding a result easier - there are no file and directory names in a database that provide hints. When choosing keywords and fields, it is important to assure that results can be identified uniquely. The above proposed fields are sufficient for the presented example, but if there would be reference calculations for AO_2, they could not be distinguished from the calculations of A: looking for the keyword "reference" and A=Cu would return both of the results. In that case one would have to either add another keyword (e.g. bulk, oxide) or add a new field (e.g. type="bulk", type="oxide") to distinguish them. Alternatively, the atoms entering the calculation could be directly analyzed.

The easiest way to visualize the relevant grouping, is to use the PHP/HTML interface which provides an intuitive way to write queries. A screen shot is shown in figure 2.5 on page 15. The query that was used is `ABO3 A=Cs` and shows all results having the keyword "ABO3" and containing a field "A" with the value "Cs". The rest of the query `columns=...` in the figure selects the columns and defines to display the atoms, that should be displayed. These argument are optional, but convenient when viewing a data set with custom fields.

Now we would like to calculate the heat of formation (hof) by finding matching calculations. To make the syntax easier we denote in the following ABO_3 as the calculation of ABO_3 and $ABO_3.PotentialEnergy$ as the potential energy of ABO_3. The hof is evaluated as

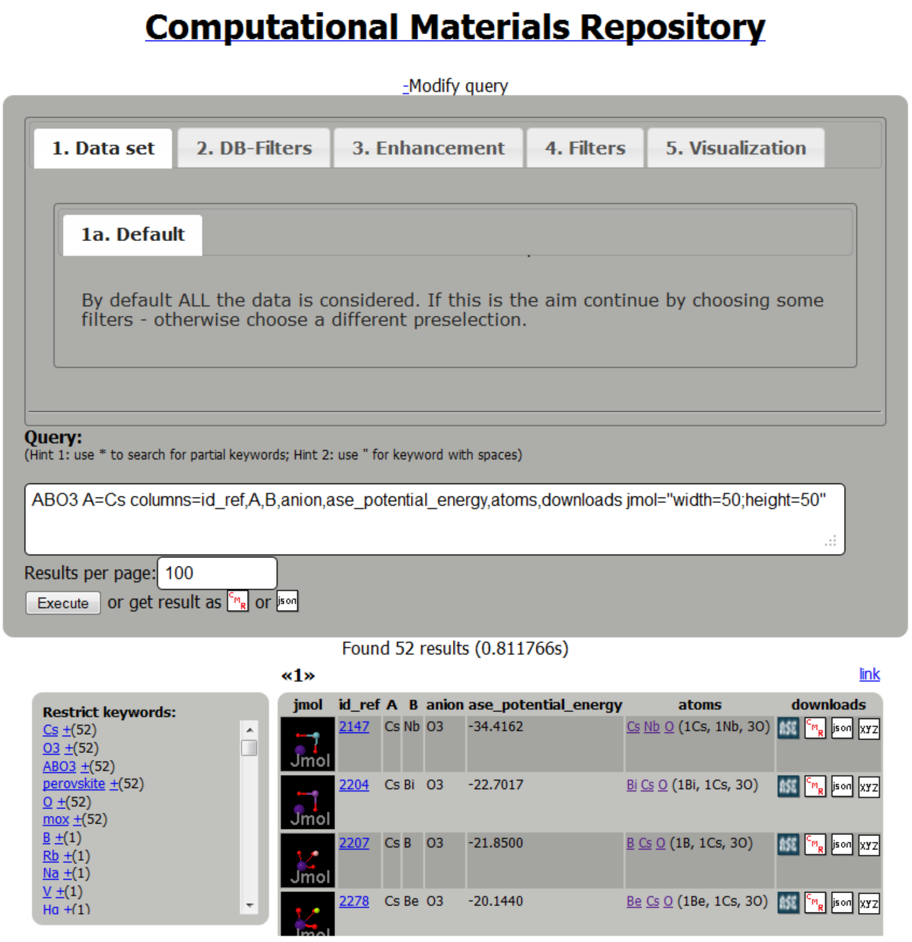
$$\begin{aligned} E_{hof} =& ABO_3.PotentialEnergy - A.PotentialEnergy/A.nA \\ & - B.PotentialEnergy/B.nA - X.PotentialEnergy \end{aligned}$$

where A.nA denotes the number of atoms of type A. We therefore need to find the four calculations in order to perform the computation.

The combination criteria can be written as three equations and be passed on to the *`GroupingAgent`*.

- $ABO_3.A = A.A$
- $ABO_3.B = B.A$
- $ABO_3.X = X.A$

The items A, B, X and ABO_3 are identified by the keywords. We also include the user name (db_user=ivca) to prevent confusions - other users might have



Computational Materials Repository

-Modify query

By default ALL the data is considered. If this is the aim continue by choosing some filters - otherwise choose a different preselection.

Query:
(Hint 1: use * to search for partial keywords; Hint 2: use " for keyword with spaces)

```
ABO3 A=Cs columns=id_ref,A,B,anion,ase_potential_energy,atoms,downloads jmol="width=50;height=50"
```

Results per page: 100

Execute or get result as CMR or json

Found 52 results (0.811766s)

«1» link

Restrict keywords:
Cs +(52)
O3 +(52)
ABO3 +(52)
perovskite +(52)
O +(52)
mox +(52)
B +(1)
Rb +(1)
Na +(1)
V +(1)
Hg +(1)

jmol	id_ref	A	B	anion	ase_potential_energy	atoms	downloads
Jmol	2147	Cs	Nb	O3	-34.4162	Cs Nb O (1Cs, 1Nb, 3O)	ase CMR json xyz
Jmol	2204	Cs	Bi	O3	-22.7017	Bi Cs O (1Bi, 1Cs, 3O)	ase CMR json xyz
Jmol	2207	Cs	B	O3	-21.8500	B Cs O (1B, 1Cs, 3O)	ase CMR json xyz
Jmol	2278	Cs	Be	O3	-20.1440	Be Cs O (1Be, 1Cs, 3O)	ase CMR json xyz

Figure 2.5: The query "ABO3 A=Cs" in the PHP/HTML interface and its result. `columns=` selects the columns that should be shown.

used the same keywords and fields for other purposes and we don't want to mix results:

A, B and X are identified with

- keyword: reference
- name-value: db_user=ivca

ABO_3 is identified with

- keyword: ABO3
- name-value: db_user=ivca

The `GroupingAgent` is a tool in the python interface that helps to group calculations. Normally agents are used to perform expensive queries and tasks in the background, but we can also make use of them directly. In our case we use one to calculate the heat of formation. The complete script that is being discussed here is shown in figure 2.6.

At the beginning, right after the imports, the definitions describe the data sets (ABO_3, A, B, X) that we need to retrieve from the database and the fields that we would like to see in the output (fields_of_interest):

```
ABO3 = {"name":"ABO3",
        "name_value": {"db_user":"ivca"},
        "keywords":["ABO3"],
        "fields_of_interest":["PotentialEnergy as e_ABO3",
                              "A",
                              "B"]}
...
```

The `name_value` (db_user=ivca) and the `keywords` (ABO3) define how ABO_3 calculations are identified. The `fields_of_interest` contains information about fields that we would like to see in the output while the `name`.*field name* is used in the definition of the combination criterion:

```
comb_crit = [("ABO3.A", "A.A"),
             ("ABO3.B", "B.A"),
             ("ABO3.X", "X.A")]
```

For every quadruple that fits the criteria, we determine the hof by writing it as an equation:

```
calc  = "ABO3.PotentialEnergy"
calc += "- ABO3.nA*A.PotentialEnergy/A.nA"
calc += "- ABO3.nB*B.PotentialEnergy/B.nA"
calc += "- X.PotentialEnergy"
```

```
from cmr.ui.db_reader import DBReader
from cmr.plugins.agents.grouping_agent import GroupingAgent
hartree = 27.211395655517308

ABO3 = {"name":"ABO3",
        "name_value": {"db_user":"ivca"},
        "keywords":["ABO3"],
        "fields_of_interest":["PotentialEnergy as e_ABO3", "A", "B"]}
A = {"name":"A",
     "name_value": {"db_user":"ivca"},
     "keywords":["references"],
     "fields_of_interest":["PotentialEnergy as e_A"]}
B = {"name":"B",
     "name_value": {"db_user":"ivca"},
     "keywords":["references"],
     "fields_of_interest":["PotentialEnergy as e_B"]}
X = {"name":"X",
     "name_value": {"db_user":"ivca"},
     "keywords":["references"],
     "fields_of_interest":["PotentialEnergy as e_X"]}

comb_crit = [("ABO3.A", "A.A"), ("ABO3.B", "B.A"), ("ABO3.X", "X.A")]

calc = "ABO3.PotentialEnergy  - ABO3.nA*A.PotentialEnergy/A.nA - ";
calc += "ABO3.nB*B.PotentialEnergy/B.nA - X.PotentialEnergy"

evaluations = {"e_heat_of_formation":"(%s)*%f" % (calc, hartree)}

keywords = ["ABO3_hof"]

items = [ABO3, A, B, X]

combination_criterions = comb_crit

agent = GroupingAgent(DBReader(),
              items,
              combination_criterions,
              evaluations,
              db_files=("", "members_only",
                        {"db_keywords":keywords}))
agent.run()
```

Figure 2.6: The calculation of the heat of formation with use of the `GroupingAgent`. (Section 2.2)

```
evaluations = {"e_heat_of_formation":"(%s)*%f" % (calc, hartree)}

keywords = ["ABO3_hof"]
```

The last step is to add some keywords to identify the results and create an instance of the `GroupingAgent` and call its run method. During the run the created db-files (results) are written directly to the db-file repository.

```
agent = GroupingAgent(DBReader(),
                      [ABO3, A, B, X],
                      combination_criterions,
                      evaluations,
                      db_files=("", "members_only",
                                {"db_keywords":keywords}))

agent.run()
```

Instead of writing the data back to the database we can print it as well onto the screen and create a spreadsheet in the csv format. The following slightly modified code shows how:

```
...
dc = DataCollection()
agent = GroupingAgent(DBReader(),
                      criterions,
                      combination_criterions,
                      evaluations,
                      data_collection = dc)

agent.run()

dc.print_table(-1, columns=["A", "B", "X", "e_heat_of_formation"])
open("all.csv", "w").write(dc.get_csv())
```

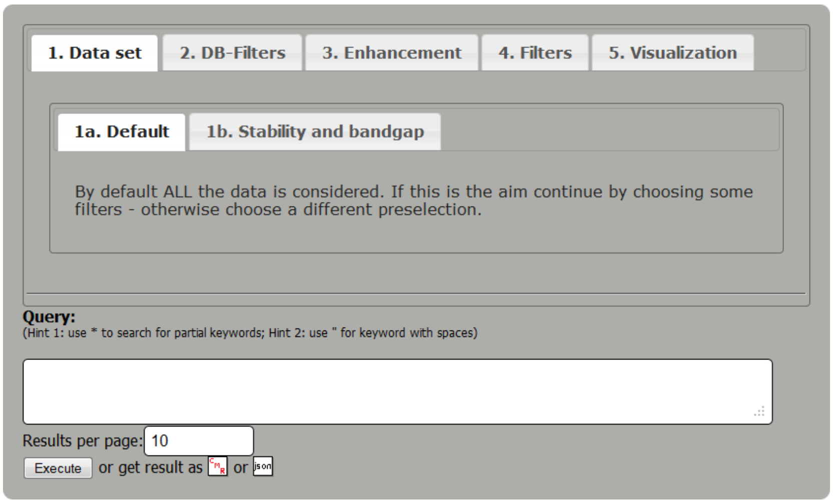
After running the agent, the `DataCollection` object contains the result. Calling `print_table` prints a table and `get_csv` returns a string in the csv format that can be written to disk. A further going introduction to the `DataCollection` will be given in the next section 2.4.

The earlier created hof results can be found with the keyword ABO3_hof. A click on the id in the PHP/HTML user interface shows the e_heat_of_formation along with with links to the calculations that were used to obtain the number. In order to create the heat map shown in figure 2.2 we open the heat map tab and choose the parameters as shown in figure 2.14 and click the "Add" button which will add the selection to the query. After pressing "Execute" the heatmap is shown.

2.3 PHP/HTML Web Interface

The PHP/HTML web interface was already mentioned, but the tool and its usage shall now be discussed beginning with the creation of a query and followed by a description of how to use navigate on the result page.

Query Creation
The whole web page is divided into the query box and the result view below as already shown in Fig. 2.5. The **query box** (Fig. 2.7) is again divided into two parts: the lower with a text box where a query can be written and the upper that actually helps to create the query. The tabs in the top row are ordered according to the expected workflow: First the data sets are selected, then the fields and keywords and atoms are restricted. As a next step the retrieved data can be enhanced which means something is added to the result. This could be for example a link or some simple mathematical operation on fields. Then filter can be applied again. The difference between the db-filters and the filters are that the db-Filters are executed by the database (which is very fast) and the filters are applied after the retrieval (slow). The reason for having these filters is that we might want to perform operations on a calculated field. As a last step we can decide how to visualize the result.


By default ALL the data is considered. If this is the aim continue by choosing some filters - otherwise choose a different preselection.

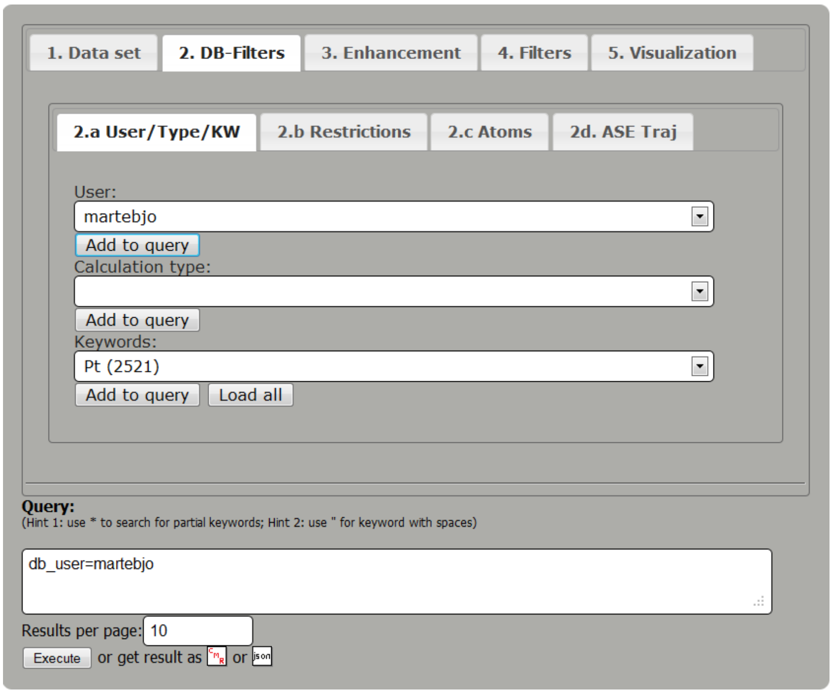
Query:
(Hint 1: use * to search for partial keywords; Hint 2: use " for keyword with spaces)

Results per page: 10

Figure 2.7: The **query box** enables to write a query in order to retrieve or search data.

In the above screenshot two data sets can be chosen: the default one which is simply all data in the database and the stability and band gap screening which limits the data set to this specific purpose. For now we choose the default which means we simply go to tab db-filter. Again we get a few sub-tabs presented: the first options to restrict by the user, by the type of calculation and by keywords:

In order to select a user we open the dropdown box that is labeled with “User”



User: martebjo

Calculation type:

Keywords: Pt (2521)

Query:
(Hint 1: use * to search for partial keywords; Hint 2: use " for keyword with spaces)

db_user=martebjo

Results per page: 10

Execute or get result as or json

Figure 2.8: Filters and restrictions that should be applied

and choose a name. When done, we press "Add to query", which will add in our case the text `db_user=martebjo` to the query. We could press "Execute" which would shows all results that the user martebjo uploaded to the database. The restriction tab allows in Fig. 2.9 to restrict the value of a field as for example `ase_potential_energy<0` and the atoms tab enables to add a statement like `atoms=Ag,Cu` which would force the results to contain both atoms Ag and Cu. The last tab labeled ASE Traj provides hints about how to work with ASE trajectory files: trajectories contain multiple steps of a calculation, but in most cases we are only interested in the last one. In this tab you find the needed keyword to select the first or last image.

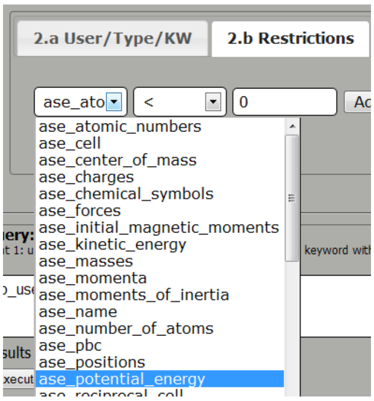


Figure 2.9: When restricting values possible fields are shown, so that one doesn't have to remember or type them.

The enhancement tab as shown in Fig. 2.10 features to add a url field, perform some simple mathematical operation and Fig. 2.11 to add a small JMol window (showing the atoms) in the table with the results.

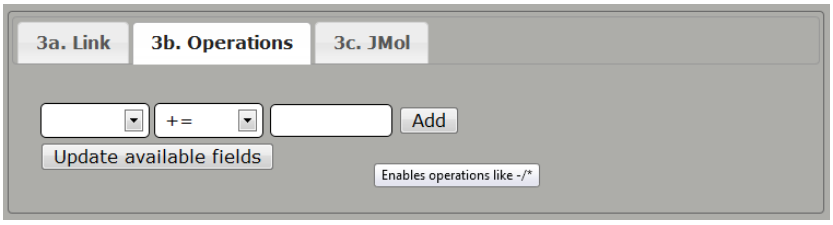
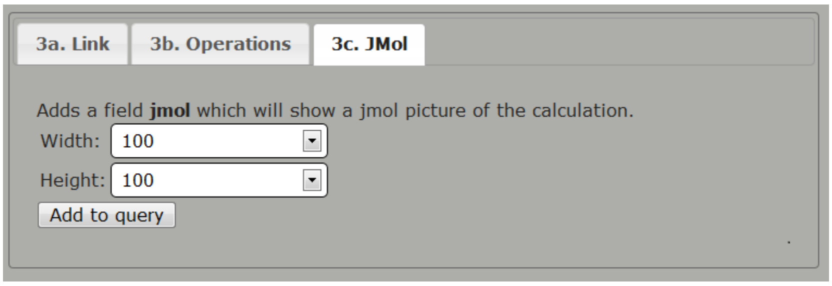


Figure 2.10: The operation sub-tab allows to perform simple mathematical operations on a field.

As a next step (4) it is possible to restrict calculated or enhanced results. This is used for instance when restricting the range of the heat of formation which is



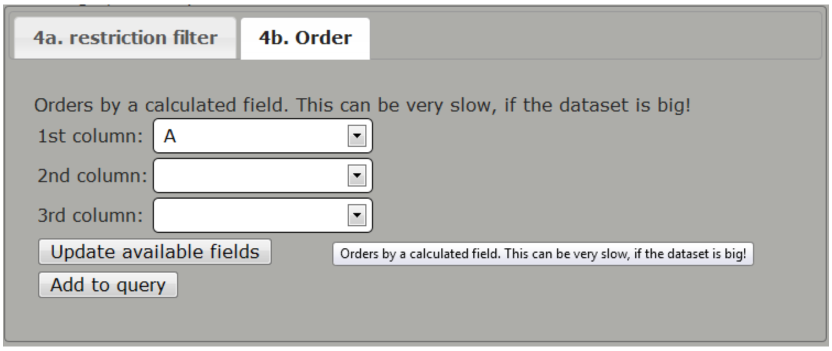
Adds a field **jmol** which will show a jmol picture of the calculation.

Width: 100

Height: 100

Figure 2.11: Add Jmol to visualize the atom in the result view.

calculated and not stored in the database. (It is also possible to restrict fields that are stored in the database but it is far less efficient than using the db-filters.) The other tab defines how to order the results.



Orders by a calculated field. This can be very slow, if the dataset is big!

1st column: A

2nd column:

3rd column:

Orders by a calculated field. This can be very slow, if the dataset is big!

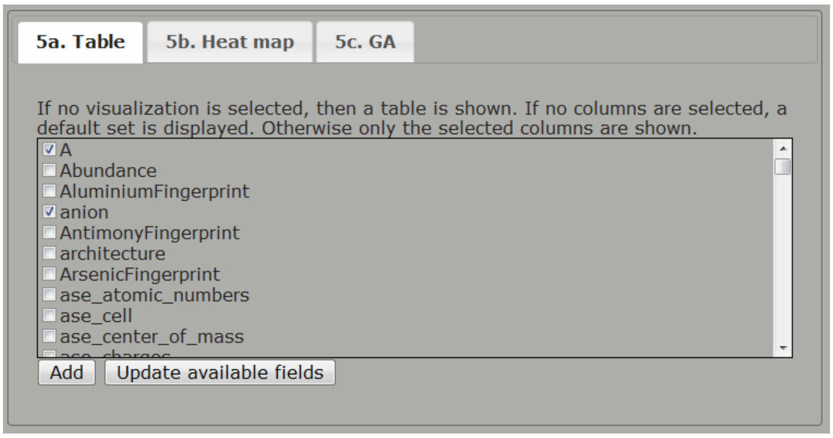
Figure 2.12: Orders the result according to the selected columns.

In the last step (5) the visualization is defined. If nothing is explicitly chosen the data is shown in a table. The displayed columns can be selected in the table subtab shown in Fig. 2.13.

The output will look similar to the screenshot in Fig. 2.5.

The Heat map tab allows to create a heat map as shown in Fig. 2.14.

The X/Y axis ticks define the field that is used on the axis and the X/Y sort order can be given a predefined order. Triangle 1 to 4 are the fields that should be shown in the heat-map (although if we choose only one field its a actually shown as a square). The colors for are defined as pairs of value and color names. e.g. 0-¿yellow defines that the color at 0 is yellow. Values between the defined colors are interpolated while values outside are extrapolated. The two special



If no visualization is selected, then a table is shown. If no columns are selected, a default set is displayed. Otherwise only the selected columns are shown.

- [x] A
- [ ] Abundance
- [ ] AluminiumFingerprint
- [x] anion
- [ ] AntimonyFingerprint
- [ ] architecture
- [ ] ArsenicFingerprint
- [ ] ase_atomic_numbers
- [ ] ase_cell
- [ ] ase_center_of_mass

Figure 2.13: By default a table is shown which can be customized by choosing the displayed columns.

"values" min and max can be used to denote the minimum and the maximum values respectively. The possible choices for the color names are shown at the bottom of the heatmap definition tab. As a last feature, the an URL column can be defined that is a field that contains a link that the user is transferred to, when clicking into the heat map. Normally this link would show the details of the calculation.

The GA tab, Fig. 2.15 allows to define how the the results of a genetic algorithm run are visualized. The genetic algorithm is discussed in the section 4.2.

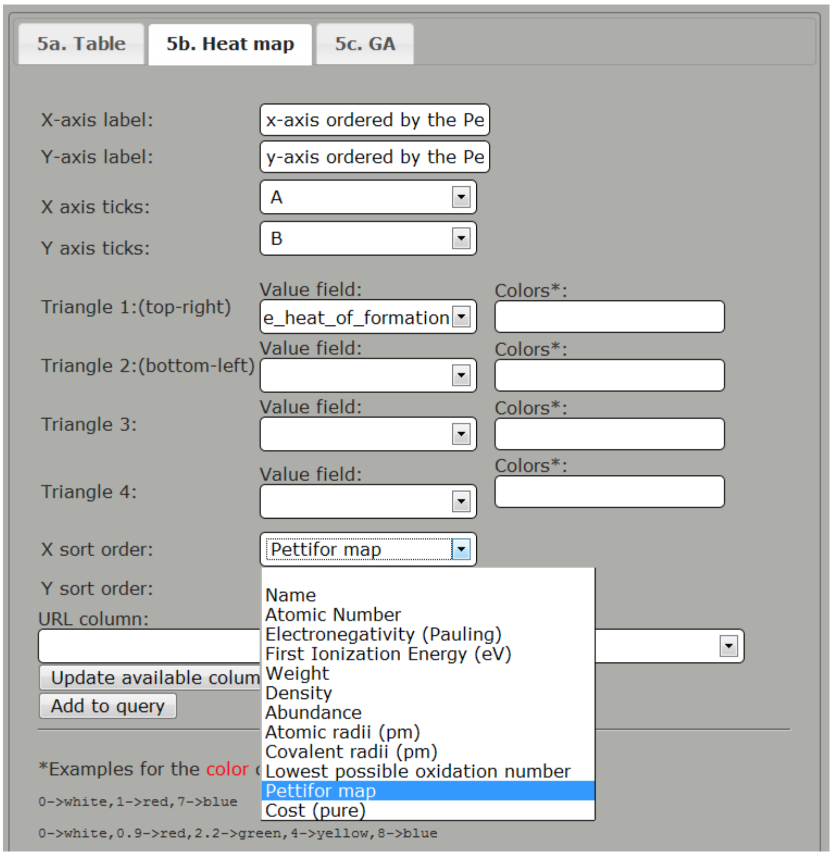
The first choice is to define whether to show the best alleles of a certain generation (best_generation) or the alleles that were calculated in that generation (generation). Then we choose the name of the run. Again it is possible to assign colors for the different values of the fitness which are assigned to the cell in the result table as shown in Fig. 2.15. The columns to display are chosen in the check box list below the other options.

Navigation

By default a table is shown with the result of the created query. The different elements are marked with numbers in Fig. 2.16 and shall be elaborated.

1 The number of results that were found. The time specifies how long it took to determine the calculations belonging to the result set. This excludes the time to actually build the web page. In some cases results are automatically cached, then the time to read the cached result is displayed.

2 The page navigation. Clicking on three will navigate to page three.



5a. Table | 5b. Heat map | 5c. GA

X-axis label: x-axis ordered by the Pe
Y-axis label: y-axis ordered by the Pe
X axis ticks: A
Y axis ticks: B
Triangle 1:(top-right) Value field: e_heat_of_formation Colors*:
Triangle 2:(bottom-left) Value field: Colors*:
Triangle 3: Value field: Colors*:
Triangle 4: Value field: Colors*:
X sort order: Pettifor map
Y sort order:
URL column:

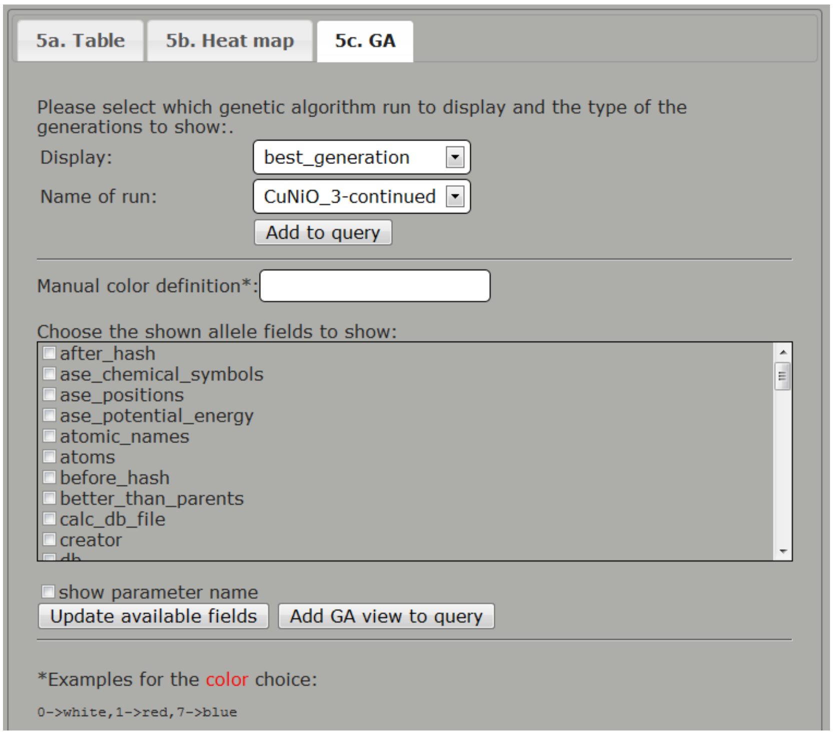
Name
Atomic Number
Electronegativity (Pauling)
First Ionization Energy (eV)
Weight
Density
Abundance
Atomic radii (pm)
Covalent radii (pm)
Lowest possible oxidation number
Pettifor map
Cost (pure)

Update available colum
Add to query

*Examples for the color

```
0->white,1->red,7->blue
0->white,0.9->red,2.2->green,4->yellow,8->blue
```

Figure 2.14: The settings to create a heat map similar to the one shown in figure 2.2



Please select which genetic algorithm run to display and the type of the generations to show:.

Display:	best_generation
Name of run:	CuNiO_3-continued
	Add to query

Manual color definition*:

Choose the shown allele fields to show:

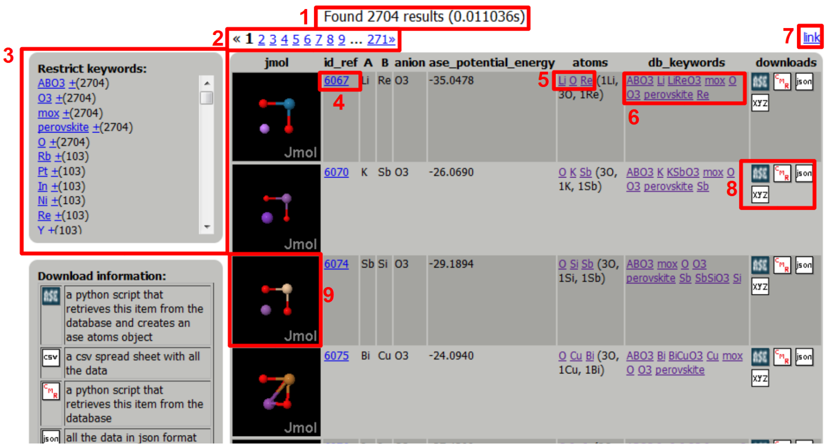
- after_hash
- ase_chemical_symbols
- ase_positions
- ase_potential_energy
- atomic_names
- atoms
- before_hash
- better_than_parents
- calc_db_file
- creator

show parameter name

*Examples for the color choice:

```
0->white,1->red,7->blue
```

Figure 2.15: Defines how to present a run of the Genetic Algorithm



Found 2704 results (0.011036s)

« 1 2 3 4 5 6 7 8 9 ... 271»

Restrict keywords:
ABO3 ±(2704)
O3 ±(2704)
mox ±(2704)
perovskite ±(2704)
O ±(2704)
Rb ±(103)
Pt ±(103)
In ±(103)
Ni ±(103)
Re ±(103)
Y ±(103)

Download information:

ASE	a python script that retrieves this item from the database and creates an ase atoms object
csv	a csv spread sheet with all the data
CMR	a python script that retrieves this item from the database
json	all the data in json format

jmol	id_ref	A	B	anion	ase_potential_energy	atoms	db_keywords	downloads
Jmol	6067	Li	Re	O3	-35.0478	Li O Re (1Li, 3O, 1Re)	ABO3 Li LiReO3 mox O O3 perovskite Re	ASE CMR json xyz
Jmol	6070	K	Sb	O3	-26.0690	O K Sb (3O, 1K, 1Sb)	ABO3 K KSbO3 mox O O3 perovskite Sb	ASE CMR json xyz
Jmol	6074	Sb	Si	O3	-29.1894	O Si Sb (3O, 1Si, 1Sb)	ABO3 mox O O3 perovskite Sb SbSiO3 Si	ASE CMR json xyz
Jmol	6075	Bi	Cu	O3	-24.0940	O Cu Bi (3O, 1Cu, 1Bi)	ABO3 Bi BiCuO3 Cu mox O O3 perovskite	ASE CMR json xyz

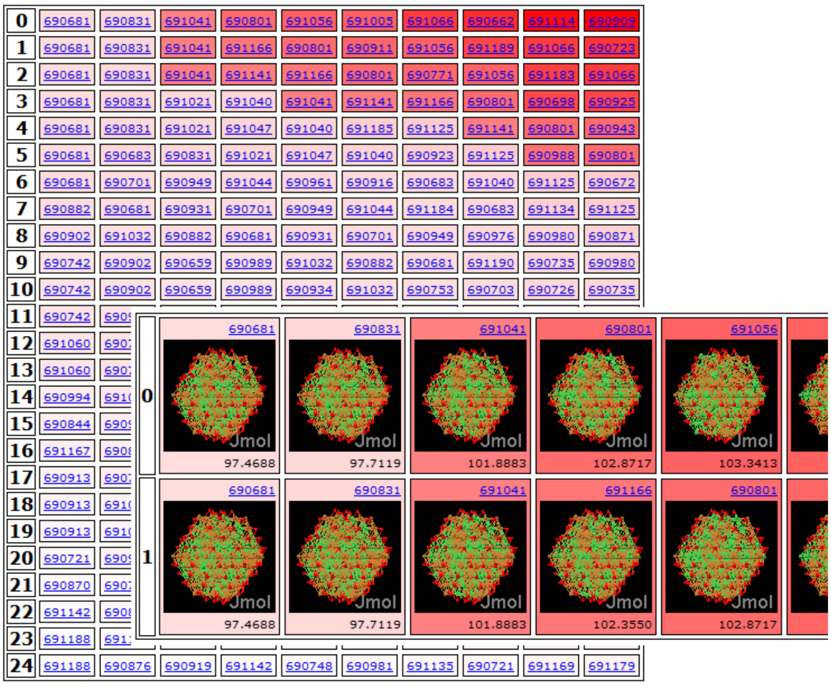
Figure 2.16: Table with the results of the query ABO3 with custom selection of the displayed columns.

3 A list of related keywords that are determined by looking at other keywords that results have in the result set. These are counted and added in descending order as suggestions. A click on the keyword will add the keyword to the query and execute it; a click on the "+" beside a keyword will add the keyword to the query, but execute it in a new window.
4 The id_ref is the id in the database. This mustn't be confused with the unique identifier presented in 2.6. A click on it will show all calculation details.
5 The atoms contained in this calculation. A click on the atom will add the atom to the query and restrict the results to calculations containing it. If the query restricts to multiple atoms only results containing all the items will be shown.
6 The keywords in this calculation. A click on them will add the selected keyword and execute it.
7 A link to the query that produced this result. It can be used to get quickly back to the presented view. Note that the query is linked and not the current results: visiting this url will show always an up to date view of the data.
8 Download links left to right: A click on the ASE icon will download a script that opens the calculation with the ASE tool "ag" which is capable of displaying the atoms and initiate a new calculation. The CMR link provides a python script template that allows to modify, delete this calculation (an example is shown in the appendix F.2), and the JSON download provides the data in JSON[18]. Last but not least an xyz[19] file can be obtained. The ASE,and xyz download are however only available for GPAW, Dacapo and ASE created files.
9 JMol view allows to look at the atom in 3D. This third-party tool needs a java plugin for the web browser.

The navigation for the results of the Genetic Algorithm and the heat map are basically the same. For example a click on the id_ref in Fig. 2.15 will open a detailed view of the clicked result.

2.4 Querying and Analysis

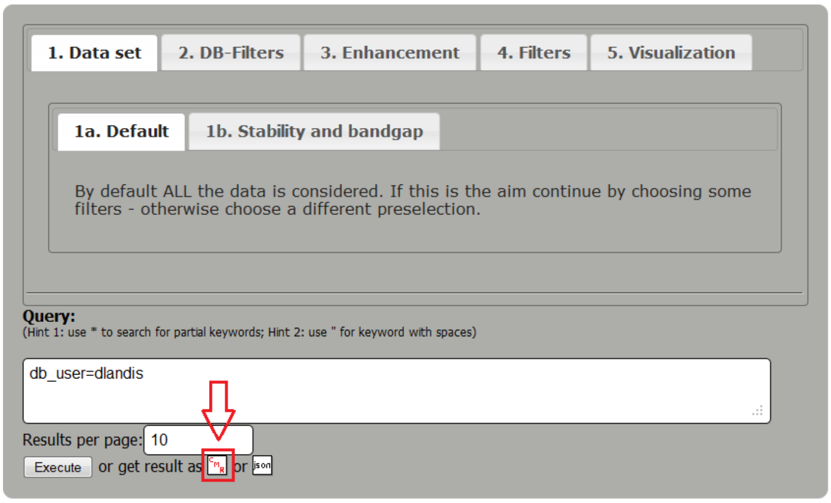
Combining the strengths of the PHP/HTML interface and the python interface makes work more efficient: the PHP/HTML interface is good for identifying and verifying data while the python interface provides the ability to filter, print, export and conduct further analysis. It is furthermore capable of connecting to multiple CMR instances, for example a CMR database at CAMD and one at an other institute in order to combine these results in some way. In this section we



0	690681	690831	691041	690801	691056	691005	691066	690662	691114	690909
1	690681	690831	691041	691166	690801	690911	691056	691189	691066	690723
2	690681	690831	691041	691141	691166	690801	690771	691056	691183	691066
3	690681	690831	691021	691040	691041	691141	691166	690801	690698	690925
4	690681	690831	691021	691047	691040	691185	691125	691141	690801	690943
5	690681	690683	690831	691021	691047	691040	690923	691125	690988	690801
6	690681	690701	690949	691044	690961	690916	690683	691040	691125	690672
7	690882	690681	690931	690701	690949	691044	691184	690683	691134	691125
8	690902	691032	690882	690681	690931	690701	690949	690976	690980	690871
9	690742	690902	690659	690989	691032	690882	690681	691190	690735	690980
10	690742	690902	690659	690989	690934	691032	690753	690703	690726	690735
11	690742	690								
12	691060	690								
13	691060	690								
14	690994	691								
15	690844	690								
16	691167	690								
17	690913	690								
18	690913	691								
19	690913	691								
20	690721	690								
21	690870	690								
22	691142	690								
23	691188	691								
24	691188	690876	690919	691142	690748	690981	691135	690721	691169	691179

0	690681 97.4688	690831 97.7119	691041 101.8883	690801 102.8717	691056 103.3413
1	690681 97.4688	690831 97.7119	691041 101.8883	691166 102.3550	690801 102.8717

Figure 2.17: Visualization of the result of a Genetic Algorithm run discussed in section 4.2. The displayed fields (fitness, jmol picture, id) are chosen in the GA tab shown in Fig. 2.15.



By default ALL the data is considered. If this is the aim continue by choosing some filters - otherwise choose a different preselection.

Query:
(Hint 1: use * to search for partial keywords; Hint 2: use " for keyword with spaces)

```
db_user=dlandis
```

Results per page: 10

Figure 2.18: CMR provides a script that executes the same query in the python interface to enable further analysis of the result and json downloads (to the right) the data in the JSON format.

look at how to work efficiently with these tools and show how to get the best performance.

We have already seen how to start a query with the `GroupingAgent`. Another way is to use a `DBReader` object with it's `find` method. We have again the possibility to select by name/value pairs, keywords and atoms. There are also choices for excluding keywords/atoms from results as can be seen in Fig. 2.19. All arguments for `find` are optional, but at least one has to be set. The return value is a derived `DataCollection` object which provides functions to create csv files, print to the screen, derive more restricted sets of data, and append `DataCollection` objects to each other.

The most efficient way to query is to make the downloaded results as small as possible with `DBReader.find`. Note that if a restriction to specific columns is omitted, `find` will download all columns of the resulting db-files. Since the download is the bottleneck you are advised to select the columns for larger amounts of data.

Commonly used operations on `DataCollection` objects are

- `DataCollection.select(name, value)` e.g. `DataCollection.select("db_user", "ivca")` returns a new collection with only items containing data from the user "ivca".

- `DataCollection.select(name, value, comparator)` e.g.
  `DataCollection.select("e_heat_of_formation", 0 , comparator=Operator("<"))` returns a new collection with only data that has $e\_heat\_of\_formation < 0$.
- `DataCollection.sort(field_name)`e.g.
  `DataCollection.sort("db_user")` sorts the collection by the user names.
- `DataCollection.select_if_in_list(name, value)` e.g.
  `DataCollection.select_if_in_list("db_keywords", ["ABO3"])` selects items from the list db_keywords that contain the keyword "ABO3".
- `DataCollection.add(other)` returns a new DataCollection instance with contains the calling and the data from the other DataCollection.

The way that the we used the `DBReader` until now just connected to the default database. In order to connect to a different one we choose a connection name:

```
#connect to default
reader = DBReader()
#connect to third party
other = DBReader("third-party")
```

If the connection is not yet known, a prompt for credentials will show up. To query the db-files that are contained in a directory we can use a `DirectoryReader` object instead of the `DBReader` (see Fig. 2.20). The limit of this approach is that `DirectoryReader`s cannot be used together with agents.

If the connection is not yet known, we are prompted for the necessary credentials.

To make the process of switching from the PHP/HTML to the python interface easier the PHP/HTML interface provides for every query a link to a python interface script that performs the same query if this is possible. (the image link highlighted in figure 2.18). The script contains also hints about how to perform common tasks such as adding keywords, fields, or removing results from the database. The idea is to use it as a starting point to retrieve the desired data for further analysis.

There are many more examples on the CMR wiki at `https://wiki.fysik.dtu.dk/cmr` that show how to work with db-files and DataCollections. The wiki's source can also be retrieved from the CMR source.

2.5 The Command Line Interface

In some situations there is need to just quickly create, modify or submit a db-file. The command line interface (CLI) enables working with db-files in a

```
import cmr
from cmr.ui import DBReader

name_op_value_list = [("PotentialEnergy","<",0.1)]
name_value_list = [("db_user", "jdoe")]
keyword_list = ["test run"]
not_keyword_list = None
not_atom_list = None
atom_list= ["Cu", "Ar"]
columns = ["PotentialEnergy", "db_user", "db_keywords"]

reader = DBReader()

collection = reader.find(name_value_list=name_value_list,
                         name_op_value_list=name_op_value_list,
                         keyword_list=keyword_list,
                         not_keyword_list=not_keyword_list,
                         atom_list=atom_list,
                         not_atom_list=not_atom_list,
                         columns=columns)

collection.print_table(20)
```

Figure 2.19: Script to retrieve data from database. The query in the PHP/HTML interface would be: `db_user=ivca "test run" atoms=Cu,Ar PotentialEnergy<0.1`.

```
import cmr
from cmr.ui import DirectoryReader

name_op_value_list = [("PotentialEnergy","<",0.1)]
name_value_list = [("db_user", "jdoe")]
keyword_list = ["test run"]
not_keyword_list = None
not_atom_list = None
atom_list= ["Cu", "Ar"]
columns = ["PotentialEnergy", "db_user", "db_keywords"]

reader = DirectoryReader("<path>")

collection = reader.find(name_value_list=name_value_list,
                         name_op_value_list=name_op_value_list,
                         keyword_list=keyword_list,
                         not_keyword_list=not_keyword_list,
                         atom_list=atom_list,
                         not_atom_list=not_atom_list,
                         columns=columns)

collection.print_table(20)
```

Figure 2.20: Script to read data/filter db-files from a directory. The query in the PHP/HTML interface would be: `db_user=ivca "test run" atoms=Cu,Ar PotentialEnergy<0.1`

shell. Additionally, the more powerful commands allow to schedule and submit results to the db-file repository similar to a file versioning tool - however there is no way to go back to older versions with CMR.

Basic Commands
Show all available commands:

```
$> cmr -h
```

To convert any supported file to the db-format `--convert` can be used:

```
$ cmr --convert inital.traj
or
$ cmr --convert *.traj
```

The first example line in the example produces the file `initial.db`, the second converts all `*.traj` in the current directory. Now the produced db-files can be enhanced with keywords, fields and attached scripts and files with the *modify* command, which will show the available sub commands:

```
$ cmr --modify inital.db
cmr version 0.3.2.523M

What would you like to do?

keywords:   ak:   add keyword(s)
            akr:  add keyword(s) recursively: also add to group members
            rk:   remove keyword(s)
            rkr:  remove keyword(s) recursively: also remove keywords
                  from group members
fields:     av:   add field/variable
            avr:  add field/variable recursively: also add to group
                  members
            rk:   remove field/variable
            rkr:  remove field/variable recursively: also remove
                  keywords from group members
scripts:    as:   add script(s)
            rs:   remove script(s)
files:      af:   add file(s)
            rf:   remove file(s)
show:       b:    browse (db-files from of "initial.db")
            s:    print status (only scheduled/added files from
                  "initial.db")
            d:    dump: dump the content of the selected items
            da:   dump-all: dump the content of the selected items
                  including all group members
commit      c:    commit: write to repository
exit        e:    exit
```

The usage of the commands is intuitive with the exception of *browse* and *print* *status* that need explaining: browse shows all selected db-files while print status considers only the status of the files scheduled to be uploaded. In our case we haven't scheduled anything and choose browse to see the a few selected columns of what is contained in `initial.db`:

```
Your choice: b

... | db_calculator        | db_user | db_keywords  |db_file_name
... | group                | cmr     | NULL         | initial.db
... | ase_trajectory_item  | cmr     | ['first']    | initial.db
... | ase_trajectory_item  | cmr     | ['last']     | initial.db
3 columns.
```

It can be seen that there are three "files": the items with the db_calculator equals "ase_trajectory_item" are the steps stored in the original trajectory file while the "group" is simply the glue that allows to reconstruct the original trajectory file. There is more information about groups in the section 2.7. In the next step we'd like to add the keywords "NEB" and "initial" to every item:

```
Your choice: akr

Enter one or more keywords (comma separated): NEB,initial
Adding keywords done.
```

The success of the operation can again be checked with `b`. Choose `e` to exit. Back on the command line the db-file can be submitted to the db-file repository with `--commit`

```
$ cmr --commit initial.db
```

Now that the file is submitted it can be found using the python interface or the PHP/HTML interface with the keywords NEB and initial.

Scheduling and Submitting

In some scenarios it is necessary to create a dataset offline and when ready upload it to the database. In case of modifications, the changed part of the dataset can be resubmitted. The commands needed to work this way are `--add` `--modify`, `--status` `--commit`. CMR then creates a subdirectory with the name `.cmr` that stores the scheduled files: don't modify them manually!

The command `--add` schedules all traj-files in the current directory as to be added to the database. During the selection the files are automatically converted to db-files.

```
$ cmr -add *.traj
cmr version 0.3.2.523

A    final.traj
A    initial.traj
```

The status can be checked with `--status`:

```
$ cmr --status .
cmr version 0.3.2.523M


status| db_orig_fn   | db_last_modified   | db_keywords| db_hash
A     | initial.traj| 2011-12-09 23:1...| NULL       | 78169b4...
A     | initial.traj| 2011-12-09 23:1...| ['first']  | 8bf505e...
A     | initial.traj| 2011-12-09 23:1...| ['last']   | eeb029a...
A     | final.traj  | 2011-12-09 23:1...| NULL       | 548209...
A     | final.traj  | 2011-12-09 23:1...| ['first']  | daf1eb2...
A     | final.traj  | 2011-12-09 23:1...| ['last']   | a2c288e...
6 columns.
```

The “A” in front signals that the files are scheduled to be uploaded. It is now possible to edit the files with `--modify` exactly the same way as previously shown:

```
$ cmr --modify initial.traj
```

with the exception that `--status` is used to see the files. When done `--commit` is used in order to upload them to the db-file repository:

```
$ cmr --commit *.traj
```

As soon as the files are available in the database, using `--status` shows the status “DB”

```
status| db_orig_fn   | db_last_modified   | db_keywords| db_hash
  DB   | initial.traj| 2011-12-09 23:1...| NULL       | 78169b4...
M DB   | initial.traj| 2011-12-09 23:1...| ['first']  | 8bf505e...
  DB   | initial.traj| 2011-12-09 23:1...| ['last']   | eeb029a...
  DB   | final.traj  | 2011-12-09 23:1...| NULL       | 548209...
  DB   | final.traj  | 2011-12-09 23:1...| ['first']  | daf1eb2...
  DB   | final.traj  | 2011-12-09 23:1...| ['last']   | a2c288e...
6 columns.
```

unless the file was modified meanwhile in which case it would be “M DB”.

The CLI has clear advantages when working directly with db-files: the access is fast and easy, but the drawback is that it is not very flexible: all actions are based on file names and are not programmable like in the python interface.

2.6 Modifying Data in the Database

There are a few important details about duplicates, overwriting and removal of data from the database. In a file system there can be only one files with a certain name per directory. CMR works in a similar way, but instead of a file name, a **unique identifier (UI)** is calculated using a hash function over the static part of the included data. The static part is mainly the output of the calculator and the dynamic part are the user added keywords, fields and attached files. This means that the UI only changes when something unexpected is modified - and prevents therefore overwriting of good with false data. There is a big advantage of having the UI calculated from the data and stored in the db-files and assigned by the MySQL database: the identifier is globally unique and as a consequence data exchange with other databases is possible without conflicts. Additionally the queries written for one database can be used with an other as well.

The right way to update a data in the database is to retrieve it, modify it and then upload it again. The assumption that converting the original file again and uploading the new db-file to the database will overwrite the existing one is wrong. The data is added, but does not overwrite anything because the created UI is different for every conversion.

In seldom cases we have to upload an "old" file and overwrite a newer one. Overwriting in the database created by CMR is similar to a file system where files are overwritten when a file with the same name is added to a directory. Normally there is an option that prevents newer files to be overwritten by older ones. This is the default in CMR: only files with a newer last modified time stamp than the one already in the database can overwrite. As a consequence when uploading older files the last modified time stamp must be updated and this happens, when the db-file is changed or if a the file is *touched*. The easiest way is to use `touch` which is available in the command line interface:

```
cmr --touch *.db
```

and also in the python interface:

```
cmr.touch(''name.db'')
# or if the file is in memory or just loaded from the database
data = cmr.read(''name.db'')
data.touch()
data.write(''name.db'')
```

After touching the db-files need to be re-uploaded.

Using the results of a colleague is easy at CAMD because there is a huge database deployed that everyone can access. Regularly people don't like to modify someone else's data - and it is also risky to not have full control over the data. In this case it is easier to *own* it. Owning means to copy the data and replace the user name with the own one. The command `own` does this and is available in the CLI:

```
cmr --own *.db
cmr --commit *.db # upload to database
```

and also in the python interface:

```
cmr.own('name.db')
# or if the file is in memory or just loaded from the database
data = cmr.read('name.db')
data.own()
data.write('.db')  # write directly to database
```

To remove files from the database the command `delete` can be used with a string list of UIs:

```
cmr.delete([string list of UIs])
```

The unique identifiers can be found in different ways depending on the location of the file. The only thing one has to know is that the unique id is called **db_hash** in db-files:

1. From a db-file with the command line interface:

   ```
   $ cmr --dump O2.db
   cmr version 0.3.2.524

   ********************************************
   Calculator/Instance: group/
   Hash: 2fff9aa2b8faaaf190ab93356cdb4d2b0acba6c3 <--
   Filename: 1.db
   Number of group members: 4
   ********************************************
   static
   ******
   db_calculator: group
   db_file_version: 0.3
   db_hash: 2fff9aa2b8faaaf190ab93356cd...         <--
   db_last_modified: 2011-12-12 12:45:47.072049
   ...
   ```

2. From random db-files in a directory:

   ```
   $cmr -b 0 . --columns db_hash,db_keywords,db_file_name
   db_hash              | db_keywords              | db_file_name
   ```

```
486f179507a31b7...  | NULL                       | ./group.db
2fff9aa2b8faaaf...  | ['N2', 'EMT']              | ./group.db
9c05fea0182582e...  | ['N', 'EMT']               | ./group.db
57e376057e28c7f...  | ['O2', 'EMT']              | ./group.db
2fff9aa2b8faaaf...  | ['N2', 'EMT']              | ./1.db
12882d67a7e6bf4...  | ['N2', 'EMT', 'first']     | ./1.db
```

Note that the number `0` in the previous command tells to align the columns automatically and the arguments following `--columns` define the column names that should be shown.

3. From a db-file repository:

```
$cmr -b 0 $CMR_REPOSITORY --columns db_hash,db_keywords,db_file_name
...
```

4. From the PHP/HTML interface: a click on *id_ref* will show all data in that calculation including the unique identifier called **db_hash**. See Fig. 2.16 item four to see where to click.

2.7 Groups

It was already shown that the conversion of an ASE trajectory file results in a group storing the contained steps and that the GroupingAgent makes groups for us. It is however often necessary to create groups manually. There are two options: The first one is to include all group members into a single db-file (Fig. 2.21) and the second one is to connect them with the UI (Fig. 2.22).

For data exchange the first approach is suitable: Only a single file needs to be sent, and it contains all data. The second approach is in general recommended especially if the data is already stored in the database.

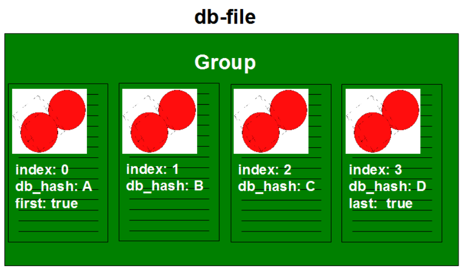


Figure 2.21: A group that contains its members in a single db-file.

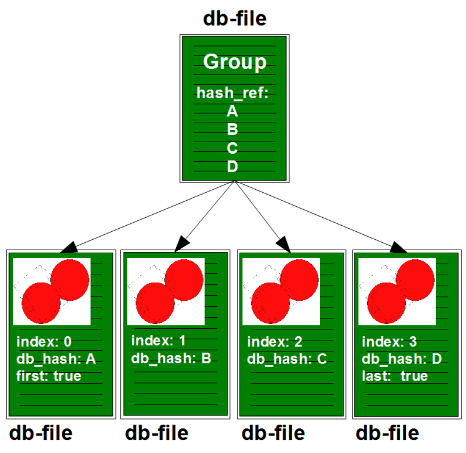


Figure 2.22: A group that references it's members. The group identifies its members by the UI which is always called **db_hash** (denoted as A,B,C,D) in the db-files.

Storing the complete members into db-files or referencing them is very similar. The first script stores them:

```
import cmr
member1 = cmr.read('1.db')
member2 = cmr.read('2.db')
member3 = cmr.read('3.db')

group = cmr.create_group()
group.add(member1)
group.add(member2)
group.add(member3)
group.write('group.db')
```

and the second one references them:

```
import cmr
member1 = cmr.read('1.db')
member2 = cmr.read('2.db')
member3 = cmr.read('3.db')

group = cmr.create_group()
group.add(member1.get_hash())
group.add(member2.get_hash())
group.add(member3.get_hash())
group.write('group_ref.db')
```

2.8 What is actually stored in a db-file?

In this section the content of a db-file shall be discussed. Note however that this section covers only the information needed by a user to work with the db-files and more details can be found in 3.3.3.b.

As shown earlier the data contained in a db-file can be viewed with the command-line interface command `-d`:

```
cmr -d N2.db
```

The content of the db-files depends on what the type of the original file was. Nevertheless there are a few fields that are always contained. Their names start with "db_" in order to prevent conflicts with possible custom fields.

field name	**sample value**
db_calculator	ase_trajectory_item
db_date	2011-12-12 12:15:54.741326
db_last_modified	2011-12-12 12:15:54.745658
db_hash	356932ea2d6bac7d6433bab8526...
db_location	CAMD Physics, DTU
db_user	dlandis
db_keywords	['EMT','N']
db_description	Calculated N2 molecule with ASE.
db_script	['N2.py']
db_files	['N2.log']

Figure 2.23: Fields that are always present in every db-file.

The meaning of the fields in Fig. 2.23 are as follows: the db_calculator field defines the original calculator, other examples besides ase_trajectory_item are gpaw, vasp, gaussian or dacapo. db_date stores the creation date of the db-file while db_last_modified the one of the last applied modification. The unique identifier is store in the field db_hash and the user name in db_user. db_location should be filled with the department and is defined when installing CMR. The more interesting field are the description (db_description), a list of keywords (db_keywords), the names of the attached scripts (db_scripts) and of other files (db_files).

For programs like gpaw and dacapo and as well ASE trajectory files the information that ASE provides is extracted and added as fields. The advantage of using them is that it we know that the units are always in eV and Angstrom.

The extracted ASE information from the calculation of a N2 molecule is shown in Fig. 2.24. This information belonging to each atom is interpreted by aligning the arrays: Fig. 2.25.

ase_atomic_numbers	[7, 7]
ase_cell	[[1.0, 0.0, 0.0], [0.0, 1.0...
ase_center_of_mass	[0.0, 0.0, 0.0]
ase_charges	[0.0, 0.0]
ase_chemical_symbols	['N', 'N']
ase_forces	[[0.0, 0.0, -0.005629824829...
ase_initial_magnetic_moments	[0.0, 0.0]
ase_kinetic_energy	0.0
ase_masses	[14.0067, 14.0067]
ase_momenta	[[0.0, 0.0, 0.0], [0.0, 0.0...
ase_moments_of_inertia	[6.9777930978192266, 6.9777...
ase_name	N2
ase_number_of_atoms	2
ase_pbc	[False, False, False]
ase_positions	[[0.0, 0.0, 0.4990868562675...
ase_potential_energy	0.262777484094
ase_reciprocal_cell	[[1.0, 0.0, 0.0], [0.0, 1.0...
ase_scaled_positions	[[0.0, 0.0, 0.4990868562675...
ase_tags	[0, 0]
ase_temperature	0.0
ase_total_energy	0.262777484094
ase_volume	1.0

Figure 2.24: These fields that are extracted with ASE and added to the db-files, if the program supports ASE.

The easiest was to read this information is actually to open the db-file with ase directly and use `view` e.g.

```
from ase.io import read
from ase.visualize import view
data = read('N2.db')
view(data)
```

or the ASE command line tool `ag`

```
ag N2.db
```

Most of the discussed content should contain sufficient information for most of the users. For more advanced usages the db-file stores also most of the fields available in the original data files with the original variable names (if possible)

	atom 1	atom 2
ase_chemical_symbols	N	N
ase_positions	[0.0, 0.0, 0.499..]	[0.0, 0.0, -0.499..]
ase_masses	14.0067	14.0067
ase_tags	0	0
...	...	...

Figure 2.25: Aligning the ASE arrays shown in Fig. 2.24 allows to identify the properties for each individual atom.

and units. In general some of the big matrices are ignored in order to keep the size of the db-file small. For GPAW all variables are stored except the following parameters: Projections, AtomicDensityMatrices, NonLocalPartOfHamiltonian, PseudoElectronDensity, PseudoPotential, PseudoWaveFunctions.

People often wonder what the name of the extracted data fields are in detail. It is out of the scope of this work to document these fields that are mostly used by code developers only, but a list of the currently extracted ones is shown in the Appendix F.1.

2.9 Advanced Task

2.9.1 Publishing results with CMR

The PHP/HTML interface can be used to share data with the public or collaborators. For certain data sets it is useful to restrict the parameters with the purpose of providing a view with specially tailored parameter choices. Examples of such a customization can be found at `http://cmr.fysik.dtu.dk`. Currently it shows the dataset of the case study described in chapter 4.1. In order to perform such a customization some knowledge in PHP and Javascript are required.

2.9.2 Using a Mapping

When working with different codes it is common that some fields have the same meaning, but a different names. In this case a mapping can be applied to an individual db-file or to a collection of db-files. (The mapping does currently not support to custom defined fields.) Fig. 2.26 shows a complete example script that retrieves some data and maps the name of TotalEnergy to e_tot and the units from eV to hartree. As shown, a dictionary defines the items that should be mapped. The mapping can then be applied to a DataCollection object.

2.9.3 Supporting a new file format

This section explains how to add support for a new file format and implement it as a CMR plug-in. Please note that this section is meant for developers and does not explain all source code in detail. Nevertheless it should be possible to create a converter by following these instructions.
In order to understand how to create a new file format you have to know that CMR supports plug-ins (Fig. 2.27). Plug-ins allow to add functionality to CMR without modifying the CMR source code. In this tutorial we are going to create a converter and a schema plug-in.

```
import os
import cmr
from cmr.ui import DirectoryReader
from cmr.tools.units import Units
from cmr.static import CALCULATOR_DACAPO
from cmr.ui.db_reader import DBReader
from cmr.base.mapping import Mapping

mapping_name="custom"

map = {
        'TotalEnergy:' :{
           "name":"TotalEnergy",         # the internal name
           "unit":Units.EV,              # the internal unit
           "type":"double",              # the internal cmr type
           "python_type":"float",        # the internal python type
           "dest_name":'e_tot',          # the new name
           "dest_unit":Units.HARTREE},   # the new unit
}

mapping = Mapping(calculator_name=CALCULATOR_DACAPO,
                  calculator_instance="",
                  name=mapping_name,
                  map=map)
cmr.definitions.register_mapping_range(mapping)

#reader = DirectoryReader(directory, "no_mapping")
reader = DBReader("default")


columns = ["db_user", "e_tot", "TotalEnergy", "atomic_numbers"]

collection=reader.find(name_value_list=[("db_user", "cmr")],
                       columns=columns)
collection.set_mapping(mapping_name) # apply mapping
collection.print_table(0, columns)
```

Figure 2.26: Example of applying a custom mapping to a DataCollection. The field TotalEnergy is mapped to e_tot, and the unit is changed from eV to hartree.

- **agent**: Agents access the database directly and can perform tasks periodically (3.3.3.a)
- **converter**: a converter (3.3.4.a) can read output files of an electronic structure code and write db-files
- **extra_parameters**: are hooks executed before the db-file is written for example during a conversion; these could be used to add extra information about the institute that performed the calculation or add data about the calculation environment or runtime information.
- **sql_schema**: defines how the data should look in the database (section 3.3.2.c CMR Database)
- **schema**: defines a cmr schema (3.3.3.c Schemas)
- **test**: a test checks if a specific functionality of CMR works and either succeeds or fails. Tests are executed after the installation or by developers to assure that modifications didn't break parts of the code. It is good practice to provide a test for every converter.

Figure 2.27: CMR plug-in types.

2.9.3.a Step 1: create a converter

We assume that we would like to create db-files from a file format that stores atoms, positions and a identifier. We call the file format newc. An example is shown here:

```
atoms = [6, 8, 8]
positions = [[0 , 0 , 0], [0 , 0 , 1.178658], [0 , 0 , -1.178658]]
name = CO2
```

The main task of a converter is to parse foreign file format and store it in a python dictionary in memory. The dictionary can then easily be written to a db-file. Fig. 2.28 shows a complete converter for the newc file format. There are three methods: accept, convert_mem and convert. accept returns true, if the file can be converted with that converter, convert_mem returns a db-file as an object while convert writes a db-file. Please note that the use of the eval function is not secure (see 3.3.3.b) and use safeeval instead. Another important remark is that we add the fields ase_positions and ase_atomic_numbers manually. The reason is that the PHP/HTML user interface (3.3.1.c) is able to show the atomic structure, if these fields are present.

```
import cmr
from cmr import logger
from cmr import Log
from cmr import CMRException
from cmr.base.converter import Converter
from cmr.tools.eval import safeeval
from cmr.io.flags import Flags

class NewC2Db:
    @staticmethod
    def accept(filename):
        """returns true, if supported by this converter"""
        return filename.endswith(".newc")

    @staticmethod
    def convert_mem(cmr_params,
                    calculator_instance="",
                    cmr_child_params=None):
        logger.log("Converting newc-file to db-file ...",
                   Log.MSG_TYPE_INFORMATION)
        filename = cmr_params["input"]

        cmr_params["db_cmr_plugin"] = "newc-plugin version 0.1"

        # create dictionary with data
        data = Converter.get_xml_writer(calculator_name="newc",
                                        calculator_instance="",
                                        mode=Flags.WRITE_MODE_CONVERT)

        lines = open(filename).read().split("\n")
        for line in lines:
            name,value = line.split("=")
            name = name.strip()
            value = value.strip()
            try:
                value = safeeval(value.strip()) #don't use eval!
            except:
                pass
            data.set_user_variable(name, value)

        data.set_user_variable("ase_atomic_numbers", data["atoms"])
        data.set_user_variable("ase_positions", data["positions"])

        return data

    @staticmethod
    def convert(cmr_params,
                calculator_instance="",
                cmr_child_params=None):
        data = NewC2Db.convert_mem(cmr_params,
                                   calculator_instance,
                                   cmr_child_params)
        data.write(cmr_params)
        data.close()
        return [data.get_hash()]
```

Figure 2.28: newc2db.py: A converter that reads *.newc files.

2.9.3.b Step 2: create a CMR schema

The next step is to create a schema (3.3.3.c). Since this task is tedious CMR offers a tool to create the main part of the schema automatically. The only thing that needs to be provided is an example of the data as a dictionary (Fig. 2.29). The next step is to create the complete schema which could look as shown in Fig. 2.30.

```
from cmr.tools.functions import make_schema

data =
{'positions': [[0, 0, 0], [0, 0, 1.178658], [0, 0, -1.178658]],
 'ase_positions': [[0, 0, 0], [0, 0, 1.178658], [0, 0, -1.178658]],
 'name': 'CO2',
 'ase_atomic_numbers': [6, 8, 8],
 'atoms': [6, 8, 8]
}
print make_schema(data)
```

Figure 2.29: Creates the main part for a CMR schema automatically.

2.9.3.c Step 3: declare the plug-ins

CMR searches plug-in directories for the file information.py where it finds the information about the class names of the available plug-ins. For our example the file is shown in Fig. 2.31

2.9.3.d Step 4: set environment variables

Add the path of the newc converter to the python path. Also adjust the environment variable CMR_SCHEMA_PATH to point to this location because we're going to create a XML schema in the next step; then add the plug-path to the cmr configuration file that is normally located in `$HOME/.cmr/cmr-settings-file` e.g.

```
plugin_paths=/home/.../newc
```

```
import os
from cmr import Units
from cmr.base.schema import Schema
from cmr.base.schema import XML_CALCULATOR

class NewCSchema(Schema):

    def __init__(self, name, hash_schema="default"):
        Schema.__init__(self, name, hash_schema=hash_schema)

        calculator = {
            "ase_atomic_numbers":{
                "cmr_name":"ase_atomic_numbers",
                "type":"long_array",
                "optional":"True",
                "python_type":"list",
                "most_inner_python_type":"int",
                "hash":True},
            "ase_positions":{
                "cmr_name":"ase_positions",
                "type":"long_x_array",
                "optional":"True",
                "python_type":"list",
                "most_inner_python_type":"int",
                "hash":True},
            "atoms":{
                "cmr_name":"atoms",
                "type":"long_array",
                "optional":"True",
                "python_type":"list",
                "most_inner_python_type":"int",
                "hash":True},
            "name":{
                "cmr_name":"name",
                "type":"string",
                "optional":"True",
                "python_type":"str",
                "hash":True},
            "positions":{
                "cmr_name":"positions",
                "type":"long_x_array",
                "optional":"True",
                "python_type":"list",
                "most_inner_python_type":"int",
                "hash":True},
        }

        self.append(XML_CALCULATOR, calculator)
        self.apply_default_values()
        self.apply_hash_schema()

    def get_my_file_name(self):
        return os.path.basename(__file__)
```

Figure 2.30: newc_03_20110523.py: Creates the main part for a CMR schema automatically.

```
from cmr import runtime

def info():
    info = [{"name":"newc2db", #module name
            "author":"cmr",
            "type":"converter",
            "classname":"NewC2Db",
            "version":"newcalc-0.3-20110523",
            "module":"newc.newc2db"},
           {"name":"newc_03_20110523", #module name
            "author":"cmr",
            "type":"cmr-schema",
            "calc_name":"newc",
            "calc_instance":"",
            "classname":"NewCSchema",
            "version":"newc-0.3-20110523",
            "module":"newc.newc"}, ]
    return info
```

Figure 2.31: information.py: declares module, class name and version of plug-ins.

2.9.3.e Step 5: create an XML schema

The xml schema allows to validate the db-file. If you made everything right, then you can load the CMR schema as follows and create the XML schema.

```
import cmr
from cmr.definitions import get_calculator_information

calc_info = get_calculator_information("newc" "")
calc_info.create_xsd()
```

Now your CMR installation supports newc files.

Chapter 3

Computational Materials Repository

3.1 Overview

The previous chapter introduced the usage of the Computational Materials Repository (CMR) from the users point of view. This chapter provides more background information about CMR with the focus on how things work and why certain design decisions were taken.

In order to investigate structures, atomization and binding energies, electron transport properties and dynamic processes, one needs to design a model that simulates the behavior at the atomic level. This requires on the one hand a fairly accurate theory to describe the interactions between the atoms and the electrons, and on the other hand, tools to place atoms and optimize their positions according to their interaction. In terms of electronic structure theory Schrödinger described in 1926 already a model that allows to determine the interaction between the atoms accurately. Unfortunately it is far too complex to be solved even on nowadays computers, because it considers all possible electron-electron interactions. Even with simplifications that treat the nuclei and the electrons separately only very small systems can be calculated. An alternative is the density functional theory (DFT) is based on the Hohenberg-Kohn theorems that explain that the many body wave function of the Schrödinger equation can be expressed with the electron density instead. Many of nowadays electronic structure simulators as for example Dacapo[20], GPAW[21] and VASP[22] are based on DFT. The most prominent result that DFT simulators provide is the total energy of a system, and the motion of the atomic positions. Depending on what type of problem is to be solve there are different ways to find solutions with electronic structure simulators. When searching for stable ternary metal borohydrides structures as performed in section 4.3 for hydrogen storage, the compounds were first tested in different structures to see which ones are potentially stable. This is done by relaxing the atoms (relaxing means that the positions of the atoms are optimized so that sum of the forces acting on each atom is minimal). In order to save computation time, all atoms but the hydrogen ones were fixed during the relaxation. The best stable compounds were then completely relaxed in order to see whether the postulated structure was the correct one or if there was a different one. Finally the compounds that allow to bind hydrogen, but not too strongly were selected as candidates and the weight percent of hydrogen was calculated. An other example is to answer the question whether or not an adsorbate would "dock" on a surface or not. In this case the surface and a few layers are modeled and the adsorbate placed on different locations on the surface. Upon comparing the DFT energies, the most favored position can be identified and it can be determined whether the adsorbate would stay. There are many other properties apart from the energy and the updated positions that can be calculated with modern electronic structure codes. An extensive introduction to electronic structure calculations and the theory behind is for example provided by [23].

This chapter starts by explaining the challenges that were faced when designing CMR. In the next step an overview of the system that we built is presented, followed by the description of each component. Then possible improvments are addressed in the outlook and the project is later compared with other implementations.

Without giving away too much details it may be helpful to know that the core of CMR is implemented in Python, is able to use a MySQL database and can provides access to the data with a web server that is programmed in PHP. The database and web server are optional for a minimal system F.3.1. CMR is usually used on a Linux system.

3.2 CMR Challenges

At the beginning there was the idea to collect data from different electronic structure simulators, and analyze them with VMDF (paper D) or an other tool. This was at the very beginning of my PhD. Very soon after, the CAMD Summer School 2008 took place where the researchers participated in a computational materials design project. The generated data had to be collected and analyzed. We were a small team that designed the workflow and I was able to gain invaluable experience by implementing the system to collect and analyze the data. The project and the implementation is outlined in section 4.3.

When designing the Computational Repository I had to find compromises between ideas, design wishes, features, resources, performance and prospects. In this section I would like to discuss the faced challenges when building CMR. The first challenge was to figure out what the expected features would be. Because CMR should handle many different kinds of problems I decided to create it generic with a focus on electronic structure simulators. This has the advantage that it can solve many different problems, but might not be optimal for some of them.

The requirements for CMR were to aggregate, store, monitor, analyze, present and share data from electronic structure simulators. Every of these requirement imposes its own challenges: aggregation requires that CMR can read the output from different codes (3.3.4.a), storing must assure that the collected data can be read also years later even if the original program is unavailable (3.3.3.b), monitoring should provide an almost instant view of the collected data, and the analysis options must satisfy power users that prefer to work with programmable interfaces (3.3.1.b) and casual users that favor more convenient access methods, like a web interface (3.3.1.c). Sharing can be performed through the interfaces (3.3.1) or by sending db-files.

In order to be able to analyze data it was decided that the user should be allowed to enhance the data with custom fields, keywords, and attach scripts and relevant files to provide more background information. Another challenge related to the analysis is the wish to be able to show what data was used to derived results (e.g. which calculations were involved when calculating the chemisorption energy). (2.7).

In terms of usability CMR must not depend on not too many third-party products and provide most of it's functionality out of the box. For example CMR must run independently of a database and a web-server which is especially important when CMR is executed on a computer cluster or on a user's computer with limited access to resources or no database access.

More important requirements are to be able to assure data consistency and prevent data loss, even if processes crash, errors occur or invalid data is tried to be uploaded (3.3.4.b). Another important issue was to find ways to provide the data with an acceptable delay.

The biggest challenge though was to provide a system that is complete and provides the expected functionality - or at least can be extended to provide it. This might seem trivial, but the way from a concept to a working implementation requires to handle a lot of technicalities, special cases and technical limitations.

3.3 System Components and Processes

The data flow of an CMR installation for an institute was already briefly presented in chapter 2. The focus in this section lies on providing more information about the components, show how they interact and explain their purpose. The data flow diagram in Fig. 3.1 highlights the three main type of components: the user interfaces (blue) that are used to interact with the data, the repositories (purple), that store the data, and the db-files (red font) that are containers for transferring and storing data. The arrows denote interactions/processes between the components.

The **conversion** (3.3.4.a)) to **db-files** (3.3.3.b) is a basic functionality of the **CMR** Python package that is accessible for other components as well. Db-files generally store the data of a single calculation and are used within CMR as a container to transfer or store data. Since db-files are identified by their content and not by their file names and directory locations they are collected in a single directory, the **db-file repository** (3.3.2.a). The data in the db-file repository has normally two purposes: it is kept for long-term storage as a backup and it should be accessible for further analysis. Since it is not very efficient to query a db-file repository, the files are uploaded to the (CMR) database (3.3.2.c). To allow queries to be executed efficiently, the data is rearranged in a better searchable way during the upload. As a consequence of this scheme the data is stored twice in the system: in the db-file repository as long-term storage and in the database for fast access. The figure shows a **local repository** that is not connected to a database as well. This is to show that CMR can also perform basic queries on the db-files as if they were located in a database - although they are a bit slower.
The user interfaces hide the complexity behind intuitive commands and functions. There are four user interfaces with different purposes: the **command line interface** (CLI) (3.3.1.a) is executed in a command shell, is the most basic interface

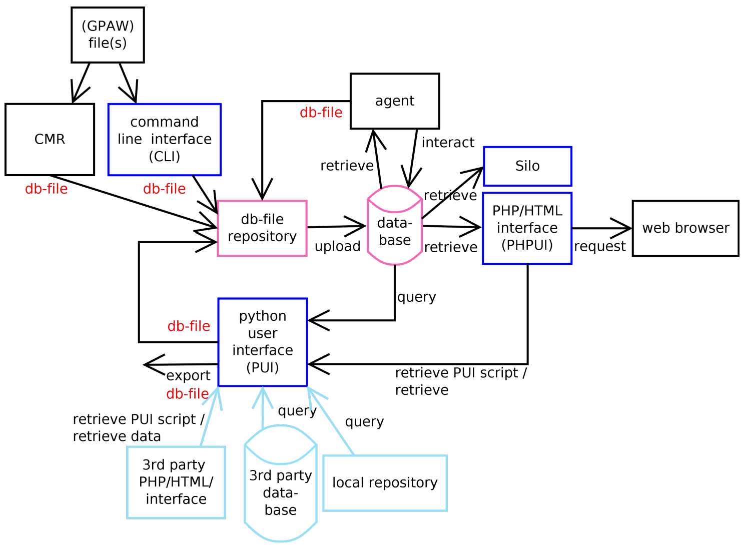


Figure 3.1: Data flow diagram of CMR. Blue: user interface components; purple are repository components; db-files are denoted with red font; black: diverse. Light blue color signifies third-party components.

and allows to create and administrate db-files locally as already described in section 3.3.1.a. The **python user interface** (PUI) (3.3.1.b) is the most powerful and provides scriptable access to the local/third-party CMR databases, is able to retrieve data from the local/third-party PHP/HTML interfaces (3.3.1.c) as well as providing basic scriptable access to a local db-file repository. The PUI can also use agent templates 3.3.3.a like the GroupingAgent (2.2), but cannot write to the database directly due to access restrictions. The most convenient access to the data is to visit the **PHP/HTML interface** (PHPUI) (2.3,3.3.1.c) with a **web browser**. The user creates a query by selecting criteria. The PHPUI can be programmed to perform more complex analysis B,3.3.1.c. Because the PHPUI is not scriptable itself it provides a PUI script that executes the same query (see Fig. 2.18) if possible. That script includes code that can be used for further analysis or to modify or download the complete data. The last interface is the HTML/JavaScript **silo user interface** (3.3.1.d) and is mentioned for completeness because it currently works only with the previous version of the CMR database. It is a workbench to analyze data in a web browser that allows to create queries visually. It is capable of working offline with the data which is not possible with the PHPUI. **Agents** (3.3.3.a) are processes that run in the background and perform tasks that normally take a little longer than regular queries. For example some analysis could be performed on all data contained in the database. Currently the agents are used to prepare data needed by the

PHPUI.

3.3.1 User Interfaces

The user interfaces enable the user to work with data in a convenient way and hide the complexity of communicating with a repository behind well defined interfaces. CMR features four user interfaces that access the data in different ways.

3.3.1.a Command line interface (CLI)

Purpose: This interface enables to work with db-files (3.3.3.b) in a command shell. It provides functionality to batch process operations like converting files to db-files, updating content, submitting data to the db-file repository and makes it thereby unnecessary to write programming code for these tasks.
Usage: The command `cmr` enables to batch process files in a command shell. There are two ways to work with this tool. The first one is to use it directly with db-files to create, modify, touch, validate, dump or submit them to the db-file repository. The second mode is to use it to work with a dataset of original output files (e.g. *.traj) that are going to be added to the CMR database. The first step is to select/schedule the files during which they are converted (transparently) to db-files. In the next steps they can be attributed with keywords/fields/script individually or as a group. When the files are ready they can be submitted. If at any later point more keywords are added they can simply be resubmitted.
Usage example(s): Scheduling all *.traj to be uploaded to the database, add keyword “adsorption” and send it to the db-file repository:

```
$> cmr -a *.traj
$> cmr -m *.traj
   ...
   ak (add keywords)
   Your choice: ak

   Enter one or more keywords (comma separated):    adsorption
$> cmr -c *.traj
```

More examples are presented in section 2.5.
Implementation: The CLI uses the CMR Python package to provide the presented functionality.
Suggested enhancements: This interface could be extended to perform queries

similar to the PHPUI (3.3.1.c) and thereby provide quicker access to results.

3.3.1.b Python User Interface (PUI)

Purpose: The PUI provides scriptable access to the databases and has access to almost all CMR functionality. This includes but is not limited to query data from the local and third-party CMR databases, download data from external CMR web servers running the PHP/HTML interface (3.3.1.c), export db-files and query data in a local repository (3.3.2.b). Last but not least it is able to use agent functionality as for example the GroupingAgent (2.2). The only thing it cannot do is write directly to the database due to access/security restrictions.
Usage: The Python user interface is available, if the CMR Python package is installed. The access to the CMR databases is provided with a special reader (DBReader). This reader allows to search and filter data by atoms, keywords and all other available fields. It is for example possible to retrieve a list of calculations with an energy smaller 0, with the keyword "adsorbate" and with the atom "C". This list can be further processed: one could just print them, add/remove keywords/fields/scripts, export the data to db-files, create a group or delete them from the database. As mentioned earlier the PUI is not allowed to perform modification on the database directly, therefore when modifying data, it is again sent to the db-file repository. The upload process will detect that the data in the database needs to be updated and perform the modifications. Removing data from the database is similar. CMR's delete function will create a db-file containing the command to remove that specific data item from the db-file repository and the database. The PUI and the DBReader can also be used to retrieve data from the web interface (PHP/HTML interface), however the unique id (UI: 2.6) must be known. The downloaded data can then be processed (add/remove keywords/fields/scripts) or exported as a db-file to a directory of choice.
The access to the third-party databases is only available if the third-party exposes them to the public, through a VPN or a ssh tunnel. Regularly this is not the case, but when it is available then the procedure and the options are exactly the same as with the local CMR database.
Additionally to the other access methods, the PUI can query any local repository that contains db-files. A special reader named DirectoryReader provides the exact same interface as the DBReader, and therefore the same functionality. The difference is that the DirectoryReader is slower because it has to load all db-files into memory before being able to execute queries.
The last access option for the PUI is to combine an agent and a DBReader. Agents can be used by regular users, as long as they don't write to the database.
Usage examples: How to use a DBReader: Fig. 2.19

How to use a DirectoryReader: Fig. 2.20
Agent: How to use a GroupingAgent Fig. 2.6
PHP/HTML interface download: Fig. 2.16, item 8 will retrieve a script that downloads the selected item from the PHP/HTML interface using the DBReader.

3.3.1.c PHP/HTML User Interface (PHPUI)

Purpose: The purpose of the PHPUI is to provide quick access to the data in the database, similar to a state of the art search engine. A query is intuitively written by adding keywords and restrictions on fields. The PHP/HTML interface allows to download data in various formats (JSON, xyz) and provides as well a script that performs the current query in the PUI (3.3.1.b).
Usage: The PHPUI is used to find and view data. Special views are available to visualize runs of the genetic algorithm (section 4.2, Fig. 2.17) or a heat map for the solar light capture (section 4.1, Fig. 1.3). The strength of this interface is that queries can be bookmarked, columns of table view chosen according to need and that the query can be composed manually or with assistance and prefilled fields as shown in section 2.3.
Usage examples: The PHPUI is accessed with a web browser. Find a keyword in the list of keywords in the query box (Fig. 2.7) and click execute. The result is displayed as in Fig. 2.16. Now the interface suggests related keywords in the box marked with (3) in the figure which can be used to restrict the results further. If necessary more filters can be added with the help of the tabs in the query box.. When the sought data (set) is found it can download as (8) xyz or JSON file, or by clicking on the ASE-link a script is provided that opens the calculation with the ASE[24] tool `ag`. When the browser is configured to execute the Python script directly then a click one the ASE icon opens `ag`[1] directly. The CMR download link provides a script that retrieves the result as a db-file or allows to modify the content of that db-file. Alternatively, if you'd like to download the whole result set or process it with the PUI, the highlighted link in Fig. 2.18 provides a PUI script enables to do so. (An example of such a script is shown in appendix F.2.
Implementation: The PHP/HTML interface runs on a web server and translates the requests from the web browser into database language and returns them as HTML back to the web browser. To avoid delays when loading the web page, the related keywords and pre-filled field names in the query box are updated using AJAX[25] (Asynchronous JavaScript and XML) a conglomerate of techniques to send asynchronous requests to the web server and update the web page in the browser dynamically upon arrival of the answer.
Technology: Special care was taken that the PHPUI uses only technology that is

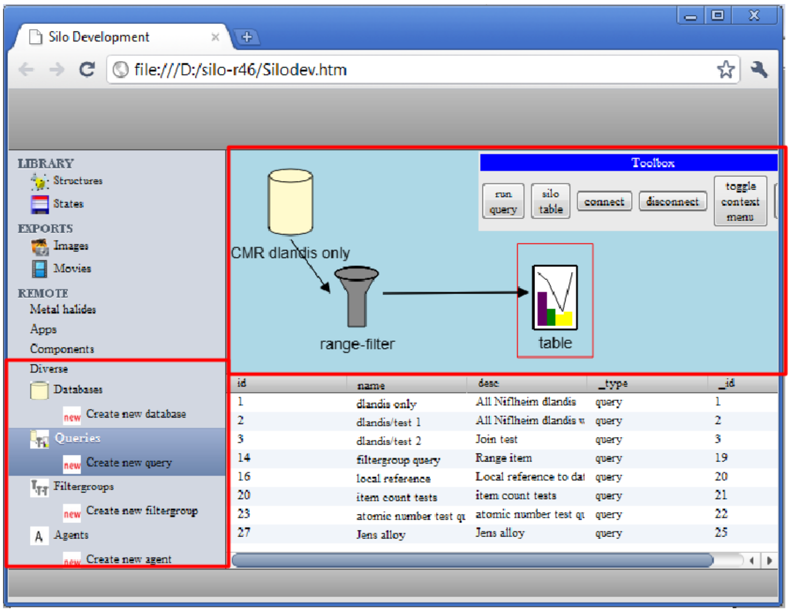
[1] `ag` is an ASE tool to visualize atomic data(-files), can calculate inter-atomic distances and create new calculations.

available on all commonly used web browsers. It is for example possible to access the web page with an Android[2] phone or a tablet computer as well. The drawback of the later two devices is however that the user interface requires a mouse for the navigation in the heat map. Security: There are three threats that need to be prevented: PHP code injection, SQL injection and cross-site scripting (XSS). The last one endangers the user only, the first two could have serious consequences for the users and the web server. PHP code injection means that an attacker is able to execute arbitrary commands on the web server. One way to achieve this is by exploiting careless evaluation of user input. Potentially dangerous functions that should not be used with unparsed and escaped input are `eval`, `preg_replace`, `include` or `require`. CMR does not use these functions. Instead it uses `doubleval` to parse double values and `strval` for strings. Preventing SQL injection is a bit trickier because the queries that comes from the PHPUI contain field names and values that are needed in the MySQL query. CMR uses two measures: input validation and escaping of all user input that goes into the query with `mysql_real_escape_string` (escaping means putting extra '\' characters in front of all character that could possibly be understood as a control character to prevent uncontrolled execution of statements.) The last measure taken is to only allow read-only access to the MySQL server from the PHPUI. XSS means to inject a script for example in a form and send it to the server. If the server does not perform any validation and returns the string unparsed it is possible for it to break out of its context and execute script code on the users web browser and thereby for example faking a password field. To prevent XSS all input that is sent back to the web browser must be encoded. PHP offers for this purpose the function `htmlspecialchars`, which is used by CMR. (Reviewed in [26, 27].)

3.3.1.d Silo Framework, and the SiGUI plug-in

The silo framework is initiated, developed and maintained by Jens Strabo Hummelshøj, at the time of writing a post-doc at SUNCAT Center for Interface Science and Catalysis at Stanford. The current implementation state of the framework and the SiGUI plug-in allows only to communicate with the previous version of the CMR database and is still in an experimental state. This framework is discussed because it is closely related to CMR, and because the author of this thesis has created SiGUI, a plug-in for visual query selection.
Purpose: The Silo web interface runs in a web browser and provides the user with tools to analyze data on his own computer, even when disconnected from the Internet. Its design allows to create, use and share plug-ins with third-parties easily. A screenshot is shown in Fig. 3.2.
Usage: With silo it is possible to administer and browse through calculations,

[2]Android is a Linux-base operating system that runs on phones and tablet computers.



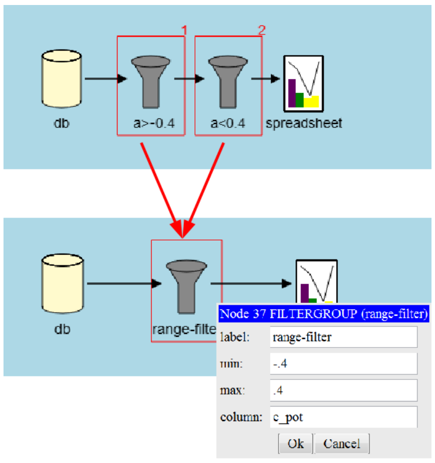
id	name	desc	_type	_id
1	dlandis only	All Niflheim dlandis	query	1
2	dlandis/test 1	All Niflheim dlandis w	query	2
3	dlandis/test 2	Join test	query	3
14	filtergroup query	Range item	query	19
16	local reference	Local reference to dat	query	20
20	item count tests	item count tests	query	21
23	atomic number test qu	atomic number test qu	query	22
27	Jens alloy	Jens alloy	query	25

Figure 3.2: Screenshot of silo and SiGUI. The red parts are created by the SiGUI plug-in (see below).

work offline, views structures and data in tables in a web browser.
Silo vs. PHPUI: Both interfaces are accessed with a web browser. The difference is where the user interface runs and in what language they are implemented: silo runs in the web browser and is mainly programmed in JavaScript while the PHPUI runs on a web server and is programmed in PHP. In terms of functionality silo is rather a desktop application while PHPUI is more a search machine.

3.3.1.d.1 SiGUI, a silo plug-in

Purpose: SiGUI (**Si**mple **G**raphical **U**ser **I**nterface) is a plug-in for Silo that was created by the author of this thesis. Its designation is to work like VMDF E and allow to create queries visually in the silo environment.
Figure: see Fig. 3.3
Usage: In order to use SiGUI silo needs to be opened in a web browser as shown in Fig. 3.2. SiGUI provides a few basic filters as for example "select" where `x op y` where x is a variable name, op an operation like >< = and y a value. New higher level filters are created by coupling a few basic filters as shown in figure 3.3 that shows that a range filter is composed of a $a > \dots$ and a $a < \dots$ filter.
Limitations: Currently SiGUI works only with the previous of the CMR database,



Node 37 FILTERGROUP (range-filter)

label:	range-filter
min:	-.4
max:	.4
column:	c_pot

Figure 3.3: `SiGUI` allows to create queries in a graphical user interface. Series of the filters like an $a > \dots$ and an $a < \dots$ can be merged into a single filter.

and created filters cannot be exchanged with third parties. The implementation level is a prototype.
SiGUI vs. VMDF: The main difference between SiGUI and VMDF is that SiGUI is implemented in JavaScript and that the filters can be composed of other filters. The main idea is to have a few general filters that can be used as a base for the creation of more complex filters which then can be used by even higher level filters. An example is shown in figure 3.3. The depicted range filter is composed of a $a > \dots$ and $a < \dots$ filter. The missing values *min* and *max* will be asked in the property window of the range-filter.

3.3.2 Repositories

The repositories are used to store data and make them accessible. There are three different types of repositories: the **db-file repository**, the **(CMR) database** and the **local repositories**. The db-file repository is the location, where users deposit the db-files that they would like to access from the CMR database later. A specially dedicated upload process (3.3.4.b) takes care of validating and copying the data to the CMR database. The local repositories are just directories that contain db-files. CMR allows to perform basic queries on the contained db-files as if they were located in a CMR database. In this section we will provide some more background information about these repositories, and how and where they are used.

3.3.2.a db-file repository

Purpose: The db-file repository acts as a data store for data that is to be analyzed or should be stored long-term.
Usage: The db-file repository is regular directory that hosts db-files and everyone (on a file system) can write his/her db-files that he/she would like to add to the CMR database. The location is identified by the environment variable `CMR_REPOSITORY`, but regularly db-files are directly written to the db-file repository by specifying the name ".db" during conversion. The conversion process is discussed in section 3.3.4.a.
Technical Details: There a few practical issues that needed to be handled for our file system. The first one is that people will submit their results with arbitrary file names and this results in file name conflicts; the second one is that storing a huge number of files results in a noticeable delay when listing the content of the directory to check for newly added db-files (100000 files took approx. 3 minutes on our NFS files server). For this reason the uploading process (3.3.4.b) renames them first and then moves them into the "inbox" which writes the file into a dedicated subdirectory if the current destination has more than 1000 files. For example the file `100059_01434...db` should be copied to the directory `inbox/`. If `inbox/` contains already 1000 files then these files are distributed into subdirectories by their first index. Our file would go to `inbox/1`. When the directory `inbox/1` has more than 1000 files it is split again, but this time the second character defines the directory. In our case the file would end up in the directory `inbox/1/0`. This pattern is followed until a directory is found with less than 1000 files. This technique is applied for all directories that store db-files within the db-file repository and assures quick access times.

3.3.2.b local repository

A local repository is a directory that contains db-files.
Purpose: Store db-files locally and perform basic queries on the db-files in that directory.
Usage: When retrieving data from third-party databases or when creating new datasets the db-files can be stored in a local repository in order to be updated/-modified before being uploaded to the database. An other use is to filter db-files without a database. `DirectoryReader` enables basic to execute basic queries and works the same way as `DBReader`. The performance is however considerably slower, because the `DirectoryReader` has to read all db-files into memory before being able filter the data.
Usage example: An example of how queries are performed on a local repository with a `DirectoryReader` is shown in Fig. 2.20.

3.3.2.c CMR Database

In this work the *CMR database* is often referred as *database*. The technical correct term would be a MySQL database with the CMR database schema.

Purpose: The CMR database allows data to be uploaded from the db-file repository and enables fast queries and downloads of data including attached scripts and files.
Usage: The upload process (3.3.4.b) loads data from the db-file repository to the CMR database. The only other component that has write access to the database is the agent (3.3.3.a). Disallowing write access to casual users ensures that they cannot break anything and that all data is uploaded consecutively.
Implementation: This section provides more information about the internal organization of the data in the database. CMR allows to use customized database schemas, but we concentrate on the default schema that is used by the PHPUI.

The first challenge is to find a **database schema** that allows storing of heterogeneous data in a relational database without knowing exactly what kind of analysis should be performed. (If we knew the analysis requirements, then we could derive an optimal table layout with standard database approaches as for example with the entity relationship model[28].)

We will first show why the the straight forward approach fails and then the CMR solution. We use a relational MySQL database (see section 3.4.3) that stores data in tables consisting of named columns and rows. The straight forward approach of storing all data in a single indexed table will not work because (A) the row-size is limited to 65535 bytes and the column-count to 4096 columns or less depending on the data that is stored in it[29] (B) adding a new column to the table means to add it to every row that is already in the database and is expensive (C) users can add arbitrary fields of arbitrary types with the same name, which will eventually result in type conflicts for the same column name. Fig. 3.4 is used to illustrate the problems.

m columns

n rows

id	Ekin	Epot	valid	...
1	.1	.01	0	...
2	.2	.02	False	...
n	...	...	...	...

Figure 3.4: A table with n rows and m columns. The upload of the second piece of data results in a conflict because there can be only one variable type per column.

(A) The row-size limit is quickly reached especially if strings are stored: if 128 bytes per string were reserved, then there would be space for 512 columns which is quickly reached considering that data from multiple simulators and an arbitrary number of custom fields can be added. (B) The set of columns cannot be determined beforehand because users can create new custom field names at any time. Every new column will result in a table reorganization that modifies all n already existing rows. This can result easily result in a delay of several minutes depending on the size of the table. (C) The above table shows a type conflict: an earlier version defined `valid` to be an integer value while the following uses boolean. In MySQL every column has an fixed type that cannot be altered. Therefore the upload with the boolean value would fail.

Since we don't know how the data will be analyzed and we cannot create a huge sparse table because the fields cannot be identified beforehand, a pragmatic approach was chosen; the variables are divided by type and written into one single table. An example is shown in Fig. 3.5. This approach results in a 5 tables, one for strings, doubles, dates, booleans, and one for arrays.

double

id	**name**	**value**
1	Ekin	.1
1	Epot	.01
1	valid	0
2	Ekin	.2
2	Epot	.02

boolean

id	**name**	**value**
2	valid	False

Figure 3.5: Example of how variables are stored in the CMR database. The data that belong together have the same id, denoted by the colors blue and green. The data is the same as in Fig. 3.4, but organized in a different way.

This schema allows fast querying, but when retrieving a whole db-file with j fields it would result in j database join operations which are expensive and slow. Therefore the db-file is uploaded and converted to a text string in the JSON[18] format which is more efficient then j joins in terms of database CPU usage. The JSON string is for example used by the PHPUI when showing a whole calculation or when downloading a whole db-file with the PUI.
Technical Details: The challenge that the processes face when querying a database schema that is organized as shown in Fig. 3.5 are that the type of the variable has to be known in order to execute the query on the right table. When the user writes the query `valid=0` how do we know in which table to look? One option is to determine the type from the input variable and then conclude that the type that is sought is the same. In this case 0 is an integer therefore the conclusion is that we look in the numeric value, but this approach fails, if we look for `surface=001`. This is because 001 is interpreted as 1 which is an integer, but

what is actually meant is the string “001”. The PHPUI considers additionally the total number of entries of “surface” in the double, string, boolean, ... tables. Regularly the result would be that “surface” is located in the string table. Since the statistical guess is weighted more than the user’s input type the choice would be correct. In some rare cases the type guess is wrong. In this case the user must either change the name of the variable, or adjust the type.
Disk Memory usage: The database schema’s memory usage is not optimal. It consumes about five to six times the amount of memory than the db-files use on disk. The reason is that all data in the db-files is compressed and in the database it is stored uncompressed. The MySQL database supports compression of columns, but unfortunately the process is not transparent. (This means the syntax to access a compressed column is different from accessing an uncompressed column.) Therefore a migration to compressed columns is difficult and implies some downtime for all CMR database installations. This issue should probably be addressed during a bigger restructuring.

3.3.3 Other Components

3.3.3.a Agents

Agents are processes that run periodically on the server and work directly on database and perform data analysis tasks or prepare information for special views or customized tables. Since users don’t have write access to the database they can generally not run agents. However some of the agents don’t write to the database and these can be used by the user.
Purpose: Agents are used to execute heavy database operations and cache data for the user interfaces. They can also be used to create customized views or to update a customized database schema.
Usage: The agents are executed periodically on the server. The frequency is defined by the developer of the agent. All registered agents (registered as a plug-in) are executed periodically. By default two agents are enabled: the first one converts the atomic numbers in the database to atomic names. The second agent creates a table that shows how often each keywords occurs and another table with the data items per user. These tables are used by the PHPUI to avoid executing these expensive queries over and over.
Usage Examples: How to use the GroupingAgent is shown in section 2.2
Background: *Software agents* can be classified according to *autonomy* (act without human interaction), *social ability* (interaction with other agents), *responsiveness* (should perceive their environment and respond to changes) and *proactiveness* (take initiative to reach goals) (reviewed in [30]). The CMR agents are currently only performing tasks autonomously and are therefore the simplest

kind of agents. The interface however doesn't prevent users from implementing smarter agents.

3.3.3.b Db-files/cmr-files

The db-files store the data of a single or multiple calculation, user-defined fields and keywords, and attached scripts and files. The content of db-files is well defined: every type of calculator gets its own CMR schema (3.3.3.c) that defines exactly what fields are stored, their type and optionally the unit. Since the data is stored in XML it is possible to validate the content against a XML schema(3.3.3.c). It is possible to create new special CMR schemas for specific purposes. For example one could create a schema for slab calculations that would ensure the user provides a field containing the number of layers and reject the db-file if it doesn't.
At the beginning we named the file(-format) db-files (database files). The choice of the name is not very advantageous though because many other products use the same name. Therefore we are about to name them cmr-files.
Purpose: Db-files were designed for data-exchange with third parties and for long-term storage. The content can be verified against a schema. The content of a db-file is as close as possible to the original file in terms of field names, types and units so that people can recognize the data in the usual format.
Usage Examples: Db-files are either created by converters (3.3.4.a), by agents (3.3.3.a) or by the PUI (3.3.1.b). The content of db-files can be viewed with the CLI (3.3.1.a) or with the PUI.
Background on long-term storage: Storing data long-term means to be able to ensure that the data remains readable. When we look at the data collected at CAMd/DTU, then the data that survived many years was most often stored in spreadsheets, without information about the original calculation - or in the the CAMd database that was maintained by H. L. Skriver[8]. Most the binary output files of calculations that were created over 4 years ago were deleted or cannot be read any more because the file-format was changed or new methods are used. Therefore we have decided to store the data in in a human readable form.

The criteria that db-files must satisfy are as follows: the content should be human and machine readable, no external libraries should be necessary to read and write them, the content should be small and verifiable. Human readable implies that people can extract data intuitively in contrast to machine readable which means that it is possible to write a preferably simple parser for the given data structure. Not using third-party software allows to be free in the design and maintenance of the file-format, enables to implement changes quickly and avoid thereby being overruled or forced to adapt due to third-party decisions. Cost for disk space is still an issue nowadays, even though the prices are constantly falling[31]: therefore db-files have to be small. In order to assure that the data is complete the db-files must be verifiable which implies that schemas can be created for them that define the content.

We decided to use XML[32, 33] (eXtended Markup Language) to represent the data and stores it in a single tar.bzip2[34, 35] compressed container. The compressed container is not only used to make the files smaller, but also because it allows to add more files as for example the script that performed the calculation and other output files. Having the script is great, when wondering how the result was obtained.

XML is human- and machine readable, widely supported and all major programming languages provide source code to parse XML data. XML can be used with XML schemas that define the content of an XML file. For example one can specify which variables are mandatory and of what variable type they have to be (e. g. integer, double, array, string). Validating a XML file against a XML schema checks if the syntax is correct and checks whether the constraints are violates. Fig. 3.6 shows an extract of data stored in the XML format.

```
...
<calculator>
  <TotalEnergy><double>-586.442</double></TotalEnergy>
  <AtomicNumbers>
    <long_array length="2">
      <long>6</long>
      <long>8</long>
    </long_array>
  </AtomicNumbers>
 ...
</calculator>
...
```

Figure 3.6: The content is intuitively obvious as follows: *TotalEnergy* is equal −586.442 and *AtomicNumbers* is an array containing 6 and 8. “double”, “long_array” and “long” are the names of the variable types.

The currently best available alternative to the db-file format is ETFS[36]. It was designed for electronic structure and crystallographic data. Since it bases on the NetCDF (Network Common DataForm) library[37, 38] it also capable of storing fields (variables), keywords and attributes like a name, a unit or even a range to give the data additional meaning. Since the implementation is in C and Fortran it is very fast. The current version 4.1.3 is fully backwards compatible according to the official web page([38]) which needs to be accentuate, because it is not a given feature. Although this format is attractive it doesn’t fit CMR because it requires third-party software.

Size of db-files: It is hard to provide an average size of a db-file because users can add arbitrary data and attach files. Therefore a few examples shall be provided of three different datasets. They are shown in Fig. 3.3.3.b.

Ternary alkali-transition metal borohydrides with Dacapo:

program	Dacapo
number of db-files	5581
total size	285 MB
average db-file size	51 KB
attached files	no

Perovskite Metal Oxides

program	GPAW
number of db-files	5621
total size	128 MB
average db-file size	20 KB
attached files	no

Automatic data collection of a students data

program	237 Dacapo, 1704 trajectory files, 358 GPAW
number of db-files	2299
total size	977 MB
average db-file size	400 KB
attached files	yes: *.py, *.traj, *.txt

Figure 3.7: Example of collected data in db-files and their average file sizes.

Security: Because db-files are used for data exchange with third parties it needs to be assured that they cannot be abused. An attacker could for example try to inject code. (This means to embed code into a db-file that is executed on the victims machine when the file is opened.) The goal could be to get passwords, modify or delete content, install a virus, et cetera. In Python all functions that interact with the operating system or allow dynamic code execution (e.g. exec, eval, os.system, os.popen, execfile, input, compile) should not be used with data from the third-party input files. The only function that CMR uses when parsing the third-party file is eval. This function is used to convert strings to numbers or arrays. Although potentially dangerous it is easy to sanitize eval by manually set the local global environment as unavailable. This prevents eval from executing any code. The small code below demonstrates the effect: first the dangerous version and then the sanitized one.

```
# dangerous
eval("open('/tmp/new', 'w').write('alert')")
# sanitized
eval("open('/tmp/new', 'w').write('alert')", {'__builtins__':None}, {})
```

For more information about eval and python in general see [16].

3.3.3.c Schemas

There are two kinds of schemas: the CMR schema and the XML schema. (The database schema is of different nature and discussed in section 3.3.2.c.) The CMR schema is the main schema and defines for every calculator (and calculator version) the fields, their types and units, if available. The XML schema is derived from the CMR schema and used in order to perform the db-file validation (3.3.4.c).
Purpose: The CMR schema is the description of the data in the db-file and is used during the extraction and during the validation.
Implementation: The CMR schema is implemented in Python and defines all variables stored in the corresponding db-file. Fig. 3.8 shows an extract of the GPAW schema. These CMR schemas can be adapted to specific program versions or they can be designed loose to accept several versions with the same schema. It is also possible to derive a schema from another one and make it specific for a certain type of calculation. One could imagine to make one for a chemisorption calculation where the atoms in the surface have to be put as separate fields.
Example: section 2.9.3

3.3.4 Processes

3.3.4.a Conversion to /writing of a db-file

The conversion process takes an original output file and converts it to a db-file.
Purpose: Create db-files and define keywords/fields and attach scripts/files.
Usage: The creation of the db-files is performed by the user with the CLI (3.3.1.a)or with the `cmr` Python package.
Process: First the file type is detected and the correct converter determined. The converter is responsible to select the correct version of the CMR schema (3.3.3.c) depending on the version of the input file. When the data is read from the original output file, all types are mapped to types that CMR supports internally (see Fig. 3.9 for a list). If an exception occurs during the conversion the partial db-file is written to disk and an error message shown to the user. The reason for writing the partial file is that DFT calculations are expensive and data should not just be lost without being inspected.
Technical Details: The converters were designed as plug-ins (2.9.3). The reason is to assure that license incompatibilities can be circumvented. CMR uses GPL. If a converter for a new file format should be created that causes a license conflict

```
class GPAWSchema(Schema):

    def __init__(self):
        ...

        calculator = {
         ...
         'PoissonStencil' :{
             "optional":False,
             "type":"string",
             "python_type":"int_or_str"},
         'XCFunctional' :{
             "optional":False,
             "type":"string",
             "python_type":"str"},
         'Epot' :{
             "optional":False,
             "unit":Units.HARTREE,
             "type":"double",
             "python_type":"float"},
         'AtomicNumbers':{
             "optional":False,
             "desc":"A list of atomic numbers.",
             "type":"long_array",
             "python_type":"numpy.array",
             "most_inner_python_type":"int"},
         'RubidiumFingerprint':{
             "optional":True,
             "type":"string",
             "python_type":"str"},
         ...
```

Figure 3.8: Extract of the GPAW's cmr-schema. `PoissonStencil`, `XCFunctional`, `AtomicNumbers` and `RubidiumFingerprint` are variables. `type` defines the internal type of the variable, `python_type` the original type, `atomic_numbers`. The field `unit` defines that Epot has the unit Hartree and the `desc` is a descriptive string of what of the value that the variable holds.

Python type	Int. type	Python	Int. Repr.
int	long_type	1	1
float	decimal_type	1.2	1.2
complex	array	1+2j	1+2j
str	string	”john”	”john”
list	array	[[1,2,3], [4,5,6]]	[[1,2,3], [4,5,6]]
tuple	array	((1,2,3), (4,5,6))	[[1,2,3], [4,5,6]]
numpy.array	array	numpy.array([])	[]

Figure 3.9: In order to enable CMR to run anywhere without third-party software, CMR uses only standard python types internally. During the conversion process or when writing fields to a db-files the types are mapped to the internal ones. The following data types are supported: boolean, int, float, complex, string, datetime.datetime, list and arrays of the previously mentioned data types. All other known data types are mapped to the internal representation.

(because of some dependencies or restrictions), it is possible to distribute the converter under a different license than GPL. The drawback is just that CMR cannot distribute this converter and the users have to install it separately.
Conversion examples: Fig. 2.4, 2.2, Fig. 2.4 and Fig. 2.3

3.3.4.b Upload

The upload process consists of three independently executed tasks: (1) moves files from the db-file repository (3.3.2.a) to an inbox directory, (2) validates the db-file and (3) uploads the data to the database (3.3.2.c).
Purpose: The upload process copies files from the db-file repository to the database.
Usage: The upload tasks are scheduled to run periodically (every 10 seconds) to check, if new files were written to the db-file repository, if so task (1) renames and moves and the files to an internal “inbox” folder. Task (2) checks the validity of the new arrivals and moves invalid files to a dedicated folder and the valid ones to the “valid” folder. Task (3) gets notified and uploads the new data to the database.
Implementation Details: Task (3) is responsible to check, if the newly arrived data is new, a duplicate, an updated or outdated data. This is determined with the unique identifier and the last modified time stamps as explained in section 2.6.

3.3.4.c Validation

Purpose: The validation of db-files assures their completeness, ensures that all required fields are present and checks the attached files for potentially dangerous file types.
Usage: The db-files are validated before they are accepted in the db-file repository (3.3.4.b). It is also possible to initiate a validation by the user. During the validation the content of the db-file is inspected: first the XML file is checked against the XML schema (3.3.3.c). It is valid, if the syntax is correct and all mandatory fields are present with the expected type. The second steps assures that all attached files are actually contained in the db-file and the last step makes sure there are no files with potentially malicious file extensions (see security).
Usage Examples: CLI: `cmr --validate`, PUI: `import cmr; cmr.validate("....")`
Security: Db-files are normally uploaded to the CMR database and all their content will be available in the PHPUI. This includes the attached scripts and files. An attacker could therefore attach a malicious file to a db-file and hope that someone executes it. To prevent this scenario, the validation rejects db-files that contain files with the following forbidden file extensions: 386, bin, cmd, com, crt, dll, exe, fon, html, js, jse, lnk, msi, msp, oxc, pif, reg, scr, sys, vbe, vbs, vs, wsc, wsf, wsh, xpi.

3.4 Tools

3.4.1 Motivation

CMR relies on a few third-party software products and technologies. In order to be useful for CMR they must be available for free[3], popular (in the sense of people like to use that product because of its advantages over competitors) and expected to be maintained and supported for a long time. Especially the expected lifetime is important because CMR is used as long-term storage for the data. Because CMR should be maintained and developed by different people, the software should be well known and commonly used. This section elaborates why Python, Apache/PHP and MySQL were selected. Products that need to be licensed for money are not discussed.

[3]Please note the distinction between free and *open-source software*[39]: free means there is no payment involved while open-source allows to study, modify and in some cases also to redistribute modified code.

3.4.2 Python

CMR must be able to read output files of different code during data aggregation, provide a scriptable user interface for finding and analyzing data, access collected and stored results long-term and communicate with the database. This can only be achieved with a programming language is popular among scientists, easy to learn, supports object oriented design, doesn't cost money and is available on most platforms and operating systems. Popular languages in science are C/C++, Java, Python or Fortran[40, 41].

We shall briefly look at the advantages/disadvantages of these languages. The fastest and most memory efficient code is produced by C/C++ and Fortran because the code is directly translated into machine language and the memory allocation/deallocation is performed by the programmer. This implies that the source code is larger and more complex than with Java and Python where a garbage collector takes care of the memory and the compiler creates Bytecode. Bytecode is an intermediate representation of the source code that is computer architecture independent, but has the drawback that it still has to be compiled which results in regularly slower execution times than binary programs. Nevertheless Python is popular among scientists due to its attractive extensions like NumPy (N-dimensional array objects), SciPy (efficient numerical routines) and matplotlib (2D and 3D plots) that simplify programming more than comparable libraries in other languages. (Reviewed in [40, 41, 42, 43, 44])

We chose Python because it fulfills the requirements and is a good compromise between ease of use and efficiency. The mentioned Python extensions are however not used by the implementation of CMR in order to keep it as independent as possible.

3.4.3 MySQL

Scalability is important for CMR. It should be able to support projects of small sizes as well as all data generated by a whole institute. To support the later it uses a database management systems often just called database. The requirements for the database software are to be able to store electronic structure calculations, search efficiently for fields and keywords, provide interfaces for Python and PHP, be easy to install and maintain and available on the many platforms.

The nowadays most commonly used type is the *relational database* which was invented by Edgar F. Codd and published in 1970[45] . The reason of the wide acceptance until now is that the concept is relatively simple: data is organized in tables with rows and columns. One row is referred to as a tuple consisting of attributes (the values in the columns). A collection of tuples with the same attributes (table) is called a relation, hence the name relational database. There are two other database types: the *NoSQL databases* (not only SQL) and the *object oriented database management systems*. The latter is capable of storing complete objects in a database and works therefore hand in hand with object oriented designed software. The NoSQL databases enjoy increased interest because they promise to solve mature shortcomings of the relational databases like the problems of scalability (relational databases are only vertically scalable[4]), throughput, high set up and maintenance cost, finding compromises between reliability and performance are addressed. (Reviewed in [46, 47, 48].)

For CMR we have decided to use MySQL, a relational database. Although the other database types have very attractive features the choice is difficult, depends on the exact usage and it is not clear how long they will be maintained which is a big drawback. The main arguments why MySQL was chosen were availability and popularity.

3.4.4 Apache/PHP

To visualize results in text and graphics CMR needs a web server, a programming language that runs on the web server and can use an interface to access/search a database and create content in *HyperText Markup Language* (HTML) that is presented to the user.

One option for the server side programming language is the general-purpose language Perl which provides a special library for web development and many other add-ons. Add-ons allow to choose the components that are needed, but make the installation on different platforms cumbersome. Java, other general-purpose language enables building a web-servers through its JSP (JavaServer Pages) and servlets, but requires quite some programming effort. Although Java and Perl are great languages they have a steeper learning curve compared to PHP (Hypertext Preprocessor) which is easy to learn, provides many build-in features such as MySQL database access, can be integrated in many web servers and widely applied. (Reviewed in [49, 50, 51]).

[4]By vertical scalable is meant upscaling: a single machine is upgraded with disk, memory, CPU in contrast to horizontally scalable which would mean adding more machines[46].

According to *Netcraft survey*[52] (in 2011) the Apache server is the most popular web server since 1996 followed by Microsoft's IIS server and nginx. nginx does not support PHP directly which results in lower performance of PHP - although the web server is generally very fast[53]. Reasons for the popularity of Apache are its feature richness: secure communication (SSL), support for server side programming (PHP languages, modules for authentication), availability on many different platforms, and that it is open-source software[54].

The choice of Apache and PHP is obvious due to their positioning in the market. They are both feature rich, exist for long, are open-source and actively developed.

3.4.5 Tools Summary

The third-party components for CMR, Python, Apache/PHP and MySQL were chosen according the requirements, popularity and availability on different platforms. Because all of these software packages are actively developed and maintained we expect them to provide a stable foundation for CMR. Since CMR is modular it would be possible to exchange third-party software and for example use a different database software.

3.5 Outlook

There are many ideas how to make CMR better in terms of increased performance, better user interfaces and improved usability. In this section the ones with the supposedly greatest effect will be shown and elaborated.

3.5.1 Improvements of upload and validation

The upload process (3.3.4.b) and validation (3.3.4.c) can be improved in several ways. During the validation and the uploading of the db-files, a log is kept about the performed actions. This log could be extended to store as well the associated file owner. As a consequence it would be possible to identify the account from which malicious or fake db-files were submitted. Another improvement that goes into the same direction is to create a process that notifies users about changes and problems during the upload process. To improve validation time of the db-file the current process could be enhanced to allow parallel validation. (Technical detail: it needs a separate process, Python 2.4 threads are not good enough.)

3.5.2 Improvements of the PHP/HTML user interface

The web interface lacks ways to directly update the data. Often it is desired to add/remove a keyword or put a description. Currently the user needs to use the provided PUI script that downloads the data, then it is modified and uploaded again. A desired solution would be to enhance the PHP/HTML user interface to permit to change keywords and custom fields directly. Another desired feature is user authentication. This would enable control of who has access to the database. Depending on how it is implemented it would also allow to restrict the accessible data. The last nice improvement would be, to create a custom user interface more easily, so that the user can adapt the view to his needs.

3.5.3 A submission tool

The public database does not allow direct data upload, and this is good due to security reasons. Nevertheless there should be a way that allows people to submit their data when they finished a paper. At CAMd we upload the data to the local database first. When the paper is published the data is retrieved and copied to the public database.

External users have however no access to the internal database. Alternative solutions are: (1) add the upload capability to the PHP/HTML interface, (2) set up an ftp server (this is a special server allows to upload/download files) and give access on request (3) implement a custom tool.

The simplest method is to use a ftp server because it allows to handle files efficiently. The customization of the PHP/HTML interface or a fully customized tools are nice to have but they are not really necessary.

3.5.4 CMR in the Cloud

Cloud computing is a service that provides resources such as memory, computing hours, hosting of databases or virtual machines. The name comes from seeing the components only through the cloud: it is unclear (and unimportant) for the subscriber what hardware is used and how it is organized exactly. Depending on whether an institute has its own computing resources and an administrator, it might be cheaper and more scalable to host the CMR database in the cloud.

3.6 Discussion

In the introduction a few projects were mentioned that collect data and present them in some way to the user. Figure 3.10 tries to categorize these systems in terms in order to make them comparable with each other. The investigated properties are briefly outlined in below and then discussed in this section:

- **Published** specifies the start of the project, if mentioned or otherwise the date of the first article when it is mentioned.
- **Open source** answers the question whether the source of the project is available
- **Web interface** is there a web interface available to allows to search for data?
- **Web tools** determines whether more advanced analysis tools are available
- **Accepts diverse file formats** declares if multiple file formats are supported as input format
- **Open for submission** tells whether users can submit data or whether the data is only created/added by the owner of the infrastructure
- **Installable** specifies whether the project be used and installed by anyone
- **Purpose** defines the main purpose of the project
- **Technology (if declared)** what is the driving technology behind the project
- **Automatic data creation** answers the question whether data is automatically created (by a smart process or if requested by the user)
- **Semantic validation** elaborates, if the plausibility of the data in some way checked?
- **Keywords/ Annotation** are keywords or annotation of data supported?
- **Reference** provides a reference (link or article)

	Materials genome	ESTEST	Quixote	Ensembl	PDB	CMR	ICSD
Published	2010-11	2010-12	2010-09 (first wiki entry)	1971	2002	2010-10 (wiki)	1990
Open source	[1]	[2]	y	y	n	y	n
Web Tools	y	n	?	y	y		y
Accepts diverse file formats	n	y	y	n	-	y	?
Open for submission	n	y	n	y	y	not yet	n
Installable	n	[2]	y	y	n	y	y (Windows)
Web interface	y	y	y	y	y	y	y (not free)
Purpose	high-throughput screening for materials	verification and validation against other electronic structure codes	collect and analyzing quantum chemical code data	find gene related information	3D structures of proteins	collect and analyze electronic structure codes	provide experimentally validated crystal structure information
Technology (if declared)	Postgresq, Java, Perl, XML	Python, XML, CGI, XQuery, eXist-db	Java, Maven, Mercurial, Avogadro, REST	?	?	Python, MySQL, XML	?
Automatic data creation	y	n	n	n	n	n	n
Semantic validation	?	y	?	data is cross-linked	y	n	y
Keywords/ Annotation	?	y	y	n	y	y	y
Reference	[10]	[9]	[12, 13]	[5]	[55]		[1]

[1] open source job control to handle convergence issues - but not other components
[2] not declaration, but password protected download link available
– probably not
? not explicitly mentioned

Figure 3.10: Properties of projects in some way similar to CMR.

A few common properties can be derived from the projects shown in Figure 3.10: Projects that handle heterogeneous file formats use an internal representation of the data (similar to the db-files). Most of them also allow to add keywords or annotations to the data (automatic or by the user), and they generally use state-of-the-art but not cutting-edge technology. The reason is probably that cutting-edge technology has a shorter expected lifetime and because of competing products might push a product out of the market. One can also see that all projects provide some sort of web interface, but only the focused projects projects are able to provide more advanced analysis tools (web toolboxes) - and the focused ones are mostly closed source. Most of the projects do not accept submissions, unless they perform validation.

There are other facts that can be derived from the above projects (not necessarily from the table though): In the physics field there are no public frameworks to analyze data. In the biology field it is possible to upload a file and have it cross-linked with already existing data. This provides a certain degree of automatic validation. Reasons why databases are not yet as elaborated for the physicists as for the biologist are that the processed data is more heterogeneous and that up to now there were no massive investments as for example for NCBI[56] in the 70s. (NCBI links Ensembl and PDB which are shown in the figure).

Where does CMR stand compared to the other physics related projects? CMR is open-source and provides a complete framework from the collection of the data to the presentation. There is no specialization to a certain type of data or process as in ESTEST, ICSD or Materials Genome. Even though CMR could possibly perform the same tasks in terms of data collection and presentation it would naturally perform worse, because of its generic nature. More overlap is found in handling file formats between Quixote, ESTEST and CMR. They are all capable of reading foreign file formats and use an internal representation or file format to store the data. Quixote is very similar to CMR. It addresses the same problems, but for quantum chemistry codes. One can see in the table, that the features are almost the same with the exception of the used technology (e.g. Java instead of Python). Still CMR has some advantages over the Quixote project: it not only allows to add keywords and fields, it also allows to add scripts and files and perform queries on the custom fields. Moreover CMR is able create arbitrary groups of data and in this way link derived data with their origin. CMR's user interface moreover allows users to access and modify their data with Python scripts and a web interface.

One can conclude from this comparison (1) that the biological databases are further in their development because they are able to link databases (2) that most of the projects have a very specific purpose and are therefore not really competing with each other and (3) that Quixote and CMR are very similar, although CMR has currently a few advantages over Quixote.

3.7 Conclusion

CMR meets the requirements to provide a framework that is able to aggregate, store, monitor, analyze, present and share data from electronic structure calculations. In order to perform data exchange, a new file format (db-files/cmr-files) was created that does not depend on third-party software and enables long-term data storage. CMR provides different kinds of user interfaces that enable users with different expertise to access data: the PHP/HTML user interface provides access within a web browser, the Python user interface allows scriptable access and the command-line interface makes working with db-files easier. The implementation works out of the box after the installation, is built from well-known stable components, is held very general and can therefore handle most of the desired tasks. However, due to the generality it doesn't work optimally in all situations. If the constraints for queries get too complicated, then a custom arrangement of the involved data in the database is required. This arrangement could be created with an agent that automatically updates it when new data is added to the database. (This agent has to be implemented by the user.) The Computational Materials Repository shows how to solve the problems related to collection, presentation and analysis of data from electronic structure codes and proofs that it works by providing a working implementation.

Chapter 4

Case Studies

In the following two sections two applications of the CMR will be discussed. The first study with a preliminary version of CMR and the second one with the current one.

4.1 Computational Screening of Perovskite Metal Oxides for Optimal Solar Light Capture

To present the data from the study below to the public and allow users to perform the analysis up to a certain extend themselves, we have created a customized version of the PHP/HTML interface (3.3.1.c). A screenshot of the result is shown in Fig. 4.1. This chapter gives a brief summary of the obtained results and will then focus on the user interface and how it interacts with the database. The user interface is accessible on `http://cmr.fysik.dtu.dk`.

4.1.1 Introduction

For a material to be able to split water based on solar light, a number of conditions have to be met[57, 58]. First of all the material of course has to be stable also when surrounded by water. Furthermore, the electronic energy gap has to be in the right range (1.5-3 eV) so that the generated electron-hole pairs have sufficient energy to perform the water splitting (for this a large gap is preferred), but at the same time the gap should be small to increase the amount of light absorbed. Further conditions are related to the positions of the band edges relative to the redox potentials for water splitting. More details are described in paper B.

This study focuses mainly on the search for stable materials with optimal well positioned bandgaps. The considered oxides and oxynitrides are in the cubic perovskite structure with the general formula ABO_3 and ABO_2N respectively, where A and B are metals. GPAW is used to calculate the DFT energy and the GLLB-SC functional[59, 60] to predict the conduction and the valence band.

As stated in the supplementary material of the paper a compound is considered non-stable if the ABO_3 energy is 0.2 eV/atom greater than the best outcome of an adopted linear programming algorithm. For ABO_3:

$$\begin{aligned}\Delta E &= \mathrm{ABO_3(s)} + \\ &- \min_{c_i}(c_1\mathrm{A\,(s)} + c_2\mathrm{B\,(s)} + \\ &+ c_3\mathrm{A_xO_y\,(s)} + c_4\mathrm{B_xO_y\,(s)} + c_5\mathrm{O})\ , \qquad (4.1)\end{aligned}$$

where A and B are the bulk metals, A_xO_y and B_xO_y are the single metal oxides included in the references and O is simply obtained from $H_2O - H_2$. The constraints for this problems are:

$$c_1 + c_3 = 1\ , \qquad c_2 + c_4 = 1\ , \qquad c_3 + c_4 + c_5 = 3\ , \qquad (4.2)$$

	ΔE [eV/atom]	Gap [eV]	Band Edges
$TlTaO_3$	0.10	2.0 (2.0)	
$GaTaO_3$	−0.03	2.1 (2.2)	
$SnTiO_3$	0.10	2.5 (2.7)	✓
$CsNbO_3$	0.18	2.8 (2.9)	✓
$AgNbO_3$ [a]	0.20	2.9 (3.5)	✓
$NaVO_3$	0.10	1.0 (1.7)	
$LiVO_3$	0.17	1.3 (2.0)	✓
$BaSnO_3$ [a]	−0.08	2.5	✓
$SrSnO_3$ [b]	0.01	2.9 (3.4)	✓
$CaSnO_3$ [b]	0.16	3.0 (3.6)	✓
$SrGeO_3$	0.16	1.2 (1.7)	✓
$CaGeO_3$	0.16	2.1 (2.7)	✓
$NaSbO_3$	0.20	1.5 (2.6)	✓

Table 4.1: Cubic perovskite oxides candidates for solar light capture materials. ΔE: formation energies per atom; gap: indirect (direct) bandgap; band edges and the check marks indicate if the band edges match with the water redox potential.

	ΔE [eV/atom]	Gap [eV]	Band Edges
$BaTaO_2N$	−0.01	2.0	✓
$SrTaO_2N$	0.00	2.1	✓
$CaTaO_2N$	0.09	2.2	✓
$MgTaO_2N$	0.19	2.1 (2.8)	✓
$PbTaO_2N$	0.19	1.9 (2.1)	
$LaTiO_2N$	0.05	2.5	✓

Table 4.2: Cubic perovskite oxynitrides candidates for solar light capture materials. ΔE: formation energies per atom; gap: indirect (direct) bandgap; band edges: the check marks indicate if the band edges match with the water redox potential.

The screening identified ten oxides and five oxynitrides with $\Delta E < 0.2$ eV/atom, $1.5 < \textit{bandgap} < 3.0$ eV and band edges that allow water splitting. The found candidates are shown in Table 4.1 and Table 4.2. Some have already been experimentally tested: $BaSnO_3$[61] and $AgNbO_3$[62]. $BaSnO_3$ splits water a little less well than $AgNbO_3$ which works in the presence of sacrificial reagents, and for the oxynitrides four of them, $BaTaO_2N$[63], $SrTaO_2N$[63], $CaTaO_2N$[63] and $LaTiO_2N$[64] perform well for hydrogene evolution.

4.1.2 Customization of the PHP/HTML user interface

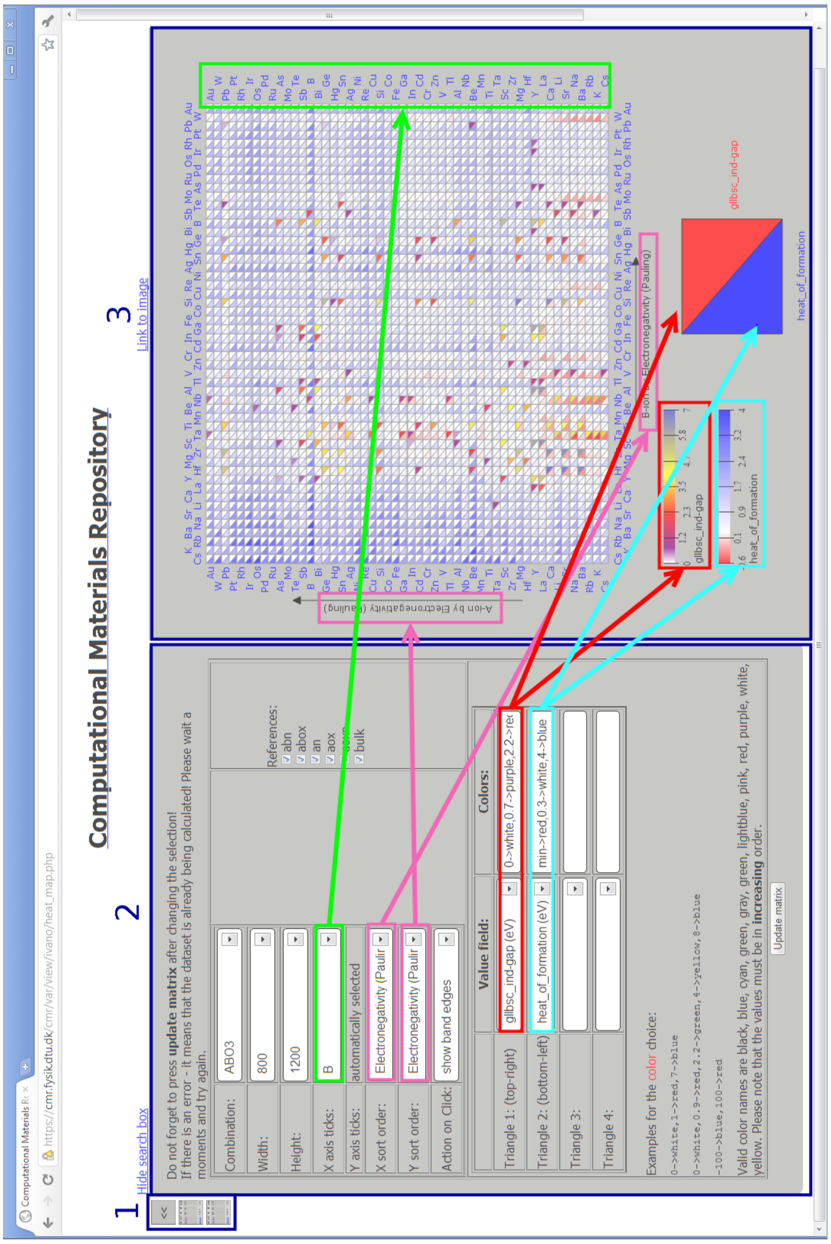
The PHP/HTML user interface (PHPUI) (3.3.1.c) is a tool to search and administrate data. The ability to write queries enables flexible searches and various choices. It requires however, that users know how to create queries and have some prior knowledge about the data that they look for. When publishing data it is however better to provide a less complicated, more intuitive and easier accessible user interface as it was implemented for this solar light capture screening. This section provides some technical background of how it was integrated into the PHPUI workflow .

4.1.2.a Views and Query

The customized user interface has three different views: one with the heat map and two to show the actual numbers of the reference and the actual data in a table. Changing between the views is done with the buttons on the left (1) shown in Fig. 4.1. The middle part (2) contains predefined fields that allows the user to choose the data set, the size, the order of the items on the axis and even the colors that should be assigned to the numbers in the heat map. When the user clicks on “Update matrix”, the choices are rewritten as a query for the PHPUI. The result is then shown on the right side (3). The query is not different from one that the user could have created with the regular PHPUI and looks as follows:

```
db_user=ivca
whiskas="combination=ABO3; refs=abn,abox,an,aox,aoxn,bulk"
heat_map="x_axis_label=B-ion by Electronegativity (Pauling);
  y_axis_label=A-ion by Electronegativity (Pauling);
  x_atom_field=B;
  y_atom_field=A;
  sort_order_x=Electronegativity (Pauling);
  sort_order_y=Electronegativity (Pauling);
  color1=0->white,0.7->purple,2.2->red,4->yellow,8->blue;
  color2=min->red,0.3->white,4->blue;
  width=800;
  height=1200;
  triangle1=gllbsc_ind-gap;
  triangle2=heat_of_formation;
  url_column=band_edge_link"
```

It looks more complicated than it is, because most of the fields are predefined. `db_user=ivca` means to choose data that was create by the user `ivca`. `whiskas` is the name of the module that is responsible to solve the linear programming problem and calculate the heat of formation. It takes two parameters: the `combination` which defines that data set (here `ABO3`) and a set of reference data


Computational Materials Repository

https://cmr.fysik.dtu.dk/cmr/var/view/ivano/heat_map.php

Do not forget to press **update matrix** after changing the selection!
If there is an error - it means that the dataset is already being calculated! Please wait a moments and try again.

Combination:	ABO3
Width:	800
Height:	1200
X axis ticks:	B
Y axis ticks:	automatically selected
X sort order:	Electronegativity (Paulir
Y sort order:	Electronegativity (Paulir
Action on Click:	show band edges

References:
☑ abn
☑ abox
☑ an
☑ aox
☑ aoxn
☑ bulk

	Value field:	Colors:
Triangle 1: (top-right)	gllbsc_ind-gap (eV)	0->white,0.7->purple,2.2->re
Triangle 2: (bottom-left)	heat_of_formation (eV)	min->red,0.3->white,4->blue
Triangle 3:		
Triangle 4:		

Examples for the color choice:

```
0->white,1->red,7->blue
0->white,0.9->red,2.2->green,4->yellow,8->blue
-100->blue,100->red
```

Valid color names are black, blue, cyan, green, gray, green, lightblue, pink, red, purple, white, yellow. Please note that the values must be in **increasing** order.

Figure 4.1: Customized PHPUI view. 1: navigation to other custom pages 2: parameters 3: the result; the arrows indicate what the parameters influence in the result window.

(here ABN, ABOx, AN, AOxN and the bulk metals) that should be used to calculate the heat of formation as shown in equation 4.1. `heat_map` takes many parameters: `x_atom_field` (x atom ticks) defines which field from the db-file is shown on the x-axis while `sort_order_x` defines their order. The available values for the sort order are always presented in a dropdown list, so the user doesn't have to memorize them. `triangle1` and `triangle2` specify which field is to be displayed in the top right respectively bottom left triangle of the squares. The color definitions `color1` and `color2` configure the color ranges for the values in the triangles. Values between the definition are interpolated.`url_column` provides the choice for the user what should be shown when clicking on a square: either a band edge plot (band_edge_link) or the raw data. The other names of the variables are. `x_axis_label` defines the label on the x-axis and the `width` and `height` the dimension of the heat map.

4.1.2.b Query Execution

The outline of the query execution is shown in Fig. 4.2. After the query is received by the PHPUI it is first parsed, which means it is split into the parts concerning the data selection, processing and visualization. In our example the data selection would be ABO3 and the user name ivca, the processing selection of the references that are to be used to determine the heat of formation and the visualization is the heat map and its paramteres. In the next step the reference data and the items from the requested data set are identified and compared with the previous runs to see if they have been calculated before. If found, then the result is loaded from the cache, otherwise the heat of formation is (re)calculated and cached before sending the result back to the user.

4.1.2.c Technical Aspects

The creation of new user interfaces can be divided into two categories: (1) query creation only and (2) query and analysis method creation. The first case is for example used to show a subset of the data and let the user choose the viewed columns or a sort order. It is then sufficient to program a user interface with PHP, HTML and JavaScript that creates a valid query and sends it to the PHPUI. In the second case when implementing a new analysis method like the linear programming it is necessary to define input parameters that can be entered in the form of a query, to process the request and read data from the database and hand the results to possibly further filters or a visualization tools. To get the linear programming working we had to retrieve the data from the database,

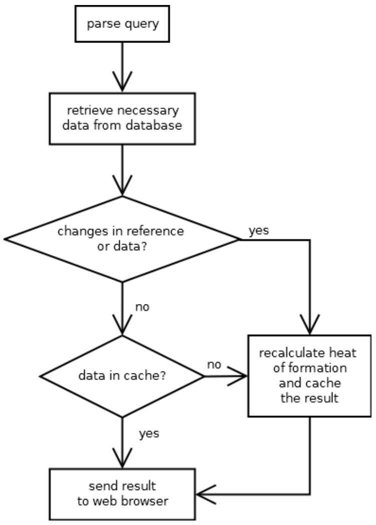

parse query
retrieve necessary data from database
changes in reference or data?
yes
no
data in cache?
no
recalculate heat of formation and cache the result
yes
send result to web browser

Figure 4.2: Simplified flow diagram through the PHP user interface.

write them to files and execute an external Python program puLP[65] to solve the linear programming problem. The output of puLP then had to be read again. The design of the caching was very important. The calculation of all the heat of formations (e.g. ABO3: 2704) takes over 90 seconds, so if the user changes the colors he/she will not want to wait to see the change.

The option to let users interactively calculate the heat of formation implies that PHP can delegate the task to solve linear programming problem to the Python program puLP. It turned out that PHP and Python are not commonly used together and therefore make the configuration of the server complicated: SELinux (Security Enhanced Linux) blocks a lot of access and functionality and had therefore be disabled. Alternative solutions would be a proper configuration of SELinux, to implement or find an alternative program that solves the linear programming problem, or to calculate the most common reference sets in advance, come up with a scheme to store these results and keep them up to date for new sets of data or references. All of these solutions are in some way cumbersome - and if possible third-party software should be avoided to be called from within PHP.

4.1.3 Conclusion

In this case study we have presented an attractive user interface that was created within the PHPUI. The user is enabled to interact with the data by applying his/her own constraint and can calculate the heat of formations with different sets of reference data on the fly. It is also explained that it is relatively easy to create a user interface that just creates a query, but that it takes more effort to provide new analysis tools. The described interfaces are available at `http://cmr.fysik.dtu.dk`.

4.2 The Genetic Algorithm with CMR

The Genetic Algorithm (GA) is inspired by natural evolution and is used to solve optimization problems as for example finding global minima. The technique is to mimic natural evolution: from a population of individuals the strong ones are mated with each other (crossed-over) or get a small modification applied (mutated). From the offspring and their parents only the fittest survive and the process restarts of mutation and crossover is applied again until the algorithm is stopped.

When combining DFT with the GA, a number of challenges are faced: DFT calculations are heavy and should therefore not be repeated. In some cases they even fail. Generally, when using the GA decision must be made about how the mutation and crossover operations should be defined. This case study presents a customizable implementation of the GA that makes use of CMR and can therefore profit from the PHP/HTML interface to visualize runs as shown in Fig. 2.17 and in Fig. 2.15. Users can build their algorithm from pre-made components, but can also add their own or replace existing ones. The features of this framework are tracing of progress with hooks, full control over the algorithm, vetoing during the creation of the new alleles and vetoing after the fitness is determined. Furthermore the calculation of the fitness can be performed in parallel on a computer cluster.

4.2.1 Introduction

After the initial description of the Genetic Algorithm (GA) by J. Holland 1975[66] it took a while until the topic gained the popularity that it has today. This is well mirrored by the number of publications per year. A quick look-up on the Web Of KnowledgeWOK[67] reveals 11′177 publications alone in 2010 (291 in physics) and looking farther back shows that the interest has steadily grown since 1990 as shown in Fig. 4.3. A reason for this interest is certainly the easy access to increased *parallel* computing power, but also that the algorithm itself is not so difficult to implement.

The GA has been previously successfully applied in physics to optimization problems as for example the prediction of the lowest energy structure of clusters[68], molecular geometry optimization[69] where they were able to find the C_{20}, C_{60} and other structures starting from random coordinates, or in the search for stable materials with specific properties as done in “Combined Electronic Structure and Evolutionary Search Approach to Materials Design”[70] where quaternary alloys were screened from 32 transition, noble and simple metals. The number

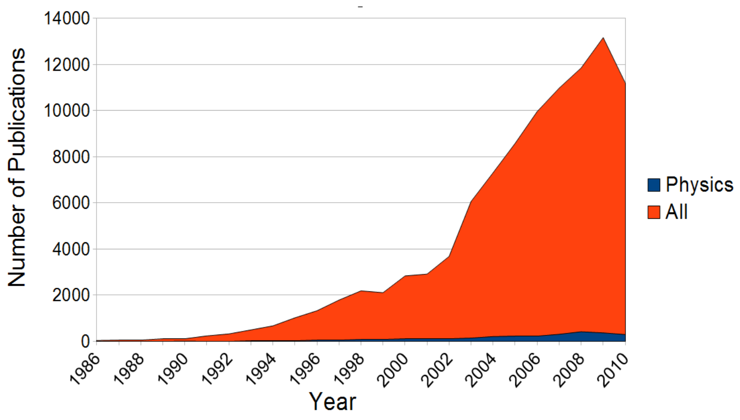


Figure 4.3: Publications with the topic “Genetic Algorithm” on the Web of Knowledge. The data about physics was retrieved by limiting the results to the subject “Physics”, 2011.08.19

of possible combinations was roughly 190′000 which made it almost impossible in 2002 to calculate all of them with the accuracy of density functional theory.

A graphical outline of the GA is shown in Fig. 4.4. There are many tunable options as for example how the individuals for the mating are chosen, how the mutation and crossovers are performed, how duplicated offspring are handled and when to stop the algorithm. To elude the algorithm and introduce common terms and definitions we look at the problem to find the most stable quaternary alloys in a *similar* way to the paper “Combined Electronic Structure and Evolutionary Search Approach to Materials Design”.

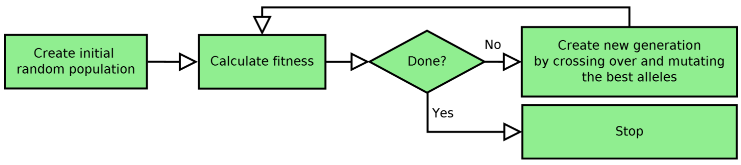

Create initial random population
Calculate fitness
Done?
No
Create new generation by crossing over and mutating the best alleles
Yes
Stop

Figure 4.4: Outline of the “Genetic Algorithm”.

The first step is to create an **initial population** consisting of randomly chosen quaternary alloys. These alloys e.g. AgIrTcTc, TcMoPtCd, ZnTiSiFe, VSiAlAl, … are called **alleles**. In order to determine which ones are good the **fitness** value must be calculated. In our case we choose the heat of formation: the lower it is the more stable (better) is the compound. The algorithm checks then whether it has to stop which is very unlikely because only one generation is

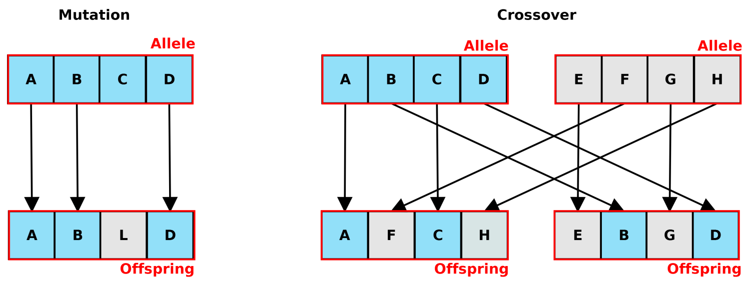


Figure 4.5: Mutation and Crossover. A, B, C, D stand for atomic names. The mutation replaces one or more atoms randomly with an other. The crossover mates two alleles. Many patterns are possible and depend on type of problem. Here half of the atoms are replaced with their pendant of the other allele.

present; instead a new **generation** is created by mutating and crossing over the good alleles from the initial population.

A **mutation** is a small modification of an **allele** as for example the exchange of one or more randomly chosen atoms e.g. the third atom in AgIrTcTc could be replaced with Rh, which would create the new **offspring** AgIrRhTc. The **crossover** chooses two individuals and mates them by exchanging approximately 50% of the atoms. Many different patterns are possible and the choice of how depends also on the atomic structure. All created offspring form the next generation. Fig. 4.5 shows an illustration of these processes. At this point the algorithm restarts the loop by determining the fitness, checking if it should stop and then again by creating the next generation.

Now that the basics of the algorithms are introduced we can focus on a few important details as for example the **population size**. As a rule of thumb[71] the size should be approximately the number of bits that are needed to represent an allele. As a simplification we assume to have 32 kinds of atoms, four positions and that the order of the atoms matters: AgIrRhTc is therefore different from AgIrTcRh. As a consequence there are $32 * 32 * 32 * 32 = (2^5)^4 = 2^{20}$ possible alloys and therefore the population size should be around 20 alleles. The **parents** to create new offspring are always taken from the last **best generation**, which is in our case the best 20 alloys. In Fig. 4.6 the subtle difference between the *generation* and *best generation* is shown: when talking about the nth **generation** we mean the created offspring in that generation, by nth *best generation* we mean the best alleles up to and including the nth generation.

As it was shown the offspring are created from the last *best generation*. There are a few different ways to **select** the parents for the mutations and crossovers. The simplest pattern is to choose them randomly. This has the advantage that

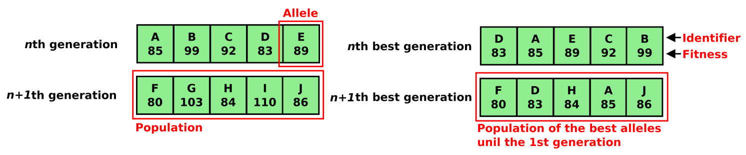


Figure 4.6: The *n+1*th *generation* contains the created offspring from the *n*th *best generation*. The *n*th *best generation* contains the best alleles from all generations up to and including the *n*th *generation*.

the population remains longer diverse, but often means that it will take long to converge. A better way is to rank the alleles and assign selection probabilities by rank. The best allele gets rank 1, the second best 2 and the n*th* gets rank n. The implementations that is discussed later provides the option to specify the selection pressure which means that the selection probabilities can be tuned to select the best alleles significantly more often than the worst, down to select all with the same probability. See Fig. 4.7 for a picture of the ranking and the assigned selection probabilities.

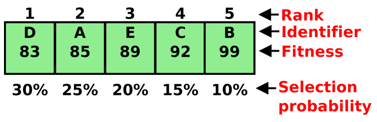


Figure 4.7: A ranking selector. The alleles are ordered according to their fitness (lower fitness is better). The better the allele, the higher the chance to be selected as a parent for offspring. In some implementations the probabilities can be tuned by a function that allows to define the selection pressure.

The last step is to define the **convergence criterion**. Again there is a vast number of possibilities. To name a few there might be a lower limit for the fitness value that has to be reached, the improvements from generation to generation are not worth the effort to continue, it might be impossible to create new offspring because the algorithm has converged, or simply because the limit of 50 generations was chosen. In our case we would stop upon convergence or when the rate of improvement becomes too little.

There are many more interesting topics, especially about mutation, crossover and selection techniques and how to make the algorithm converge quicker for example by estimating the distribution algorithm[72]. To get more insight it is recommended to read Colin R. Reeves more complete introduction to the basics of the GA in chapter five of his book “Handbook of Metaheuristics”[73].

In the consecutive sections a generic modular object oriented implementation of the GA in python is described that is tightly bound to CMR and provides therefore access to all CMR features. Originally it was written to support a screening for a project that investigated metal-oxides for light harvesting, but the implementation of the algorithm is generic enough to solve other problems. This is shown with the examples presented later. The source code is available from CAMD, DTU and the interested reader can contact Karsten W. Jacobsen to get the location for the retrieval and the usage conditions.

4.2.2 GA Components

In the previous section we have seen how the genetic algorithm works and what problems it can solve. Since DFT calculations are expensive it is natural to keep intermediate results for later data analysis, data-mining and possibly for identification of trends. We are now going to look into the implementation to see what features are necessary to solve problems. In order to follow this section the reader should know the basics of object oriented programming (OOP) - for example that classes contain fields (variables) and methods and can be derived from other classes - and that abstract classes define the interfaces and basic methods necessary to interact with other classes while concrete classes implement the required interface methods.

This implementation of the GA is generic. This means it is very flexible and can find solutions for problems formulated with numbers, vectors or atoms - and even more. In figurative speech the algorithm to solve a specific problem is tailored together the way one would build a house: one needs rooms, windows, doors, pipes et cetera. All houses are built from these **abstract types of components**. The type of house is defined by the **concrete** choices: triple glass windows and security doors are most likely used for a store and not for a family house. Fig. 4.8 shows a (not complete) list of components to build different kinds of houses.

Similar to the house, the presented GA implementation consists of abstract and concrete components (classes). Depending on which **concrete** classes are chosen, the algorithm solves a different kind of problem. Fig. 4.9 shows all available components in a **class diagram**[74]. The class names written in bold are **concrete** classes while the others are **abstract**. We shall now describe the purpose of each the components shown in class diagram of the GA.

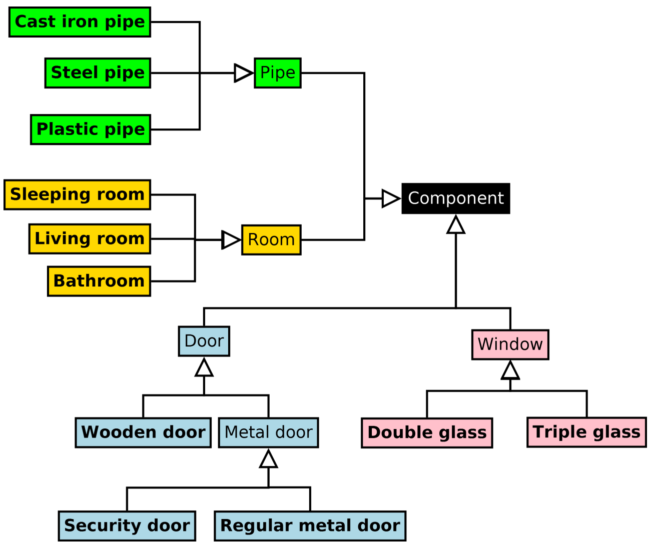


Figure 4.8: All houses are made of pipes, rooms, doors and windows, but the combination of many concrete choices define whether it is going to be a store (security door) or a family house (wooden door). The diagram should be read as follows: Security and regular metal doors are kinds of metal doors. Metal and wooden doors are a kind of doors and a door is a component. Or Double glass windows and triple glass windows are a kind of window and a window is a component. The items in bold writing are **concrete** components while the others are just an **abstract** description.

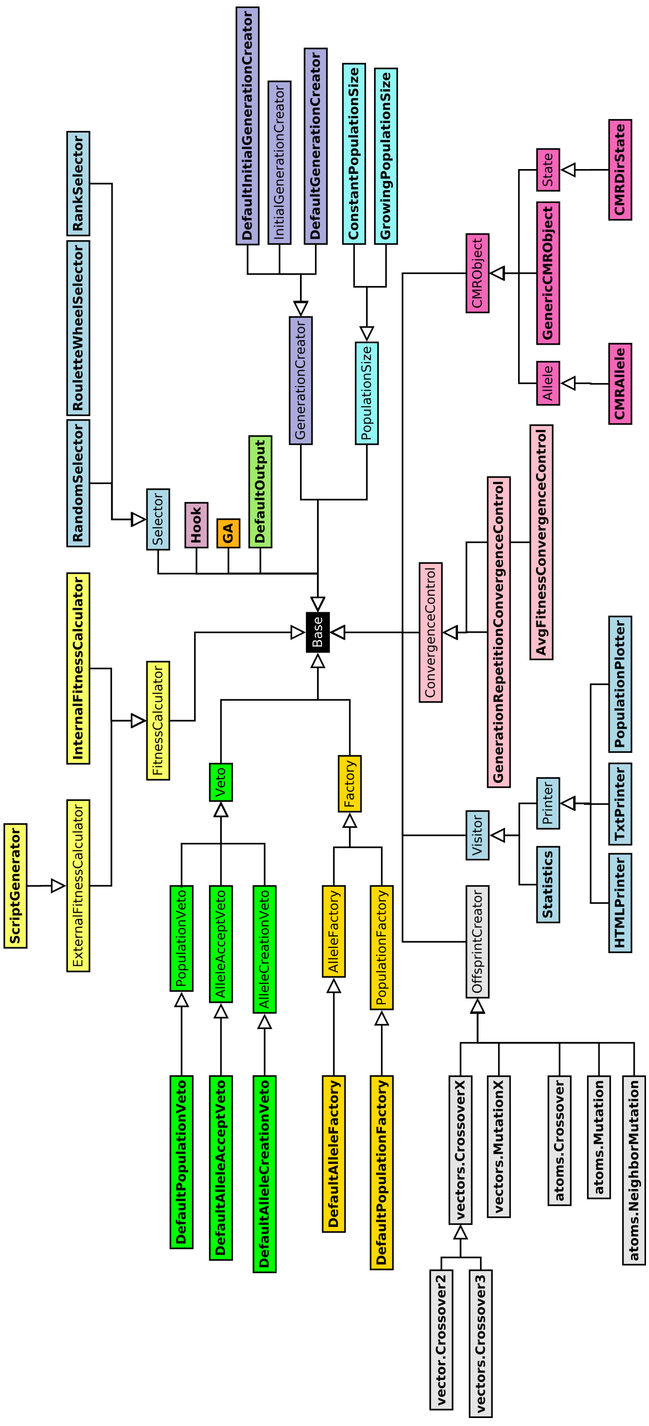


Figure 4.9: Complete class diagram for the implementation of the genetic algorithm. Boxes denote a class. **Bold** names indicate concrete classes while regular print indicates abstract classes.

4.2.2.a GA

GA is the central class that controls the algorithm as shown in Figure 4.18. The algorithm will be discussed after the components were introduced.

4.2.2.b OffspringCreator

The abstract class `OffspringCreator` is used to create offspring. The implementation supports two types of alleles: vector and atom based ones. The vector mutation class `vectors.MutationX` creates a new offspring by adding a (small) number in a specified range to every number in the vector while crossover class `vectors.CrossoverX` takes n parents and calculates an average vector. The classes `vectors.Crossover2` and `vectors.Crossover3` are convenience classes with $n = 2$ respectively $n = 3$ parents. The mutation class `atoms.Mutation` and the crossovers class `atoms.Crossover` work in the same way as the vectors, but instead of numbers the atoms are chosen from a predefined pool for the mutation. New is the class `NeighborMutation` which mutates a specified number of atoms with a neighboring atom in the periodic table. A class diagram for the offspring creation is shown in Fig. 4.10.

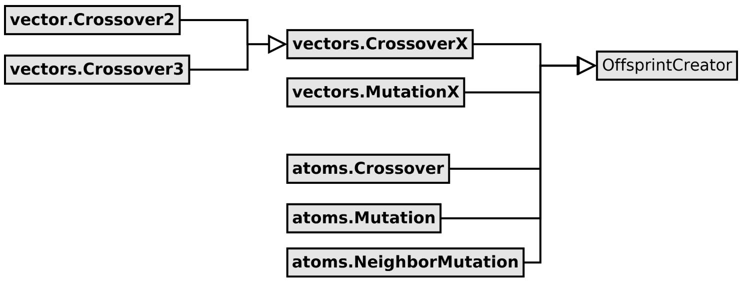


Figure 4.10: All classes that create offspring inherit from `OffspringCreator`. The `vectors.*` work with vectors while the `atoms.*` work with atoms. `vector.Crossover2` and `vector.Crossover3` are convenient classes that create offspring from 2 respectively 3 parents.

If the user chooses to implement his/her own offspring class he/she would derive it from `OffspringCreator` and add the required methods.

Generally one would create a mutation and crossover instance and define a mutation-crossover ratio. In our case a ration of 1:2 means 1 offspring is created from a mutation and 2 offspring from the crossover. The process is organized so that seen on all generations the ratio of offspring created from mutations to the ones from crossovers is 1:2. This is done when using the class `DefaultGenerationCreator`. Often though it is desired to be more flexible and perform more than just one single type of mutation and one single type of crossover. Therefore the class `GeneralGenerationCreator` allows to specify a list of mutation instances and how often then should occur.

4.2.2.c Selector

Selectors choose the parents for the offspring creation. For a mutation one and for a crossover two or more parents have to selected. The random selection (`RandomSelector`) and the ranked selection (`RankSelector`) have already been explained earlier. The `RouletteWheelSelector` can only be used to find maxima: the selection probabilities are assigned according the fitness of an allele: $p_i = \frac{f_i}{\sum f_i}$, where p_i is the seletion probability and f_i the fitness of ith allele. (See Fig. 4.11)

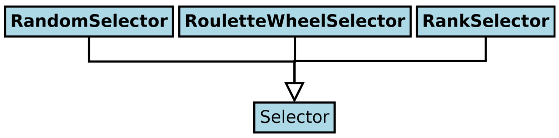


Figure 4.11: Three different selectors are supported: random selection, the roulette wheel selection and the ranking selector.

4.2.2.d Veto

For the efficiency of the algorithm it is important to eliminate duplicates as early as possible. They don't break the algorithm, but they slow it down if they occur too frequently. There are two scenarios when duplicate alleles can appear: during offspring creation or after the fitness calculation. The first case happens because the selection of the parents is done with selection probabilities and it is very likely that the same mutation or crossover is performed multiple times. The second case might shows up after the fitness is calculated. For instance when working with atoms and two different offspring are relaxed it is possible that the atoms end up in very similar positions. In both cases the duplicate alleles need to be removed or marked as duplicate. To identify the alleles two different hash values (signatures) are created for each allele: the "before" hash includes information that is available just after creation as for example atomic names and the positions. The "after" hash includes the information available after the calculation of the fitness as for example the atomic names and the relaxed atomic positions.

In this implementation the check for duplicates is modeled as veto. The veto can forbid an allele to be used as an offspring or mark it as invalid/duplicate. There are three concrete classes that use the “before” and “after” hash to veto alleles: the `DefaultAlleleCreationVeto` prevents duplicates offspring creation, `DefaultAlleleAcceptVeto` marks duplicate result alleles as failed after the fitness calculation, and `DefaultPopulationVeto` is a convenience class that checks a whole population after the determination of the fitness for duplicates at once. If a user wants to implement a different kind of veto he/she can inherit from the corresponding abstract class and implement the required methods. The involved classes are shown in Fig. 4.12.

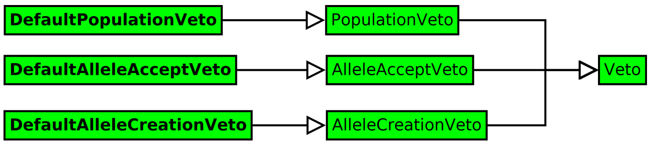


Figure 4.12: There are three different kinds of concrete veto classes provided that prevent duplication of alleless: `DefaultAlleleCreationVeto`, `DefaultAlleleAcceptVeto` and `DefaultPopulationVeto`. If the user wishes to extend the competence of the default vetoes he/she can inherit from the corresponding abstract classes (`PopulationVeto`, `AlleleAcceptVeto`, `AlleleCreationVeto`) and implement his/her own veto.

4.2.2.e FitnessCalculator

The `InternalFitnessCalculator` asks each `Allele` to calculate its fitness value consecutively. If the fitness is a DFT energy however it is more efficient to calculate it in parallel by using the `ScriptGenerator` which adapts a template calculation by adding the allele’s properties (e.g. atoms) and then submitting it to the queuing system. The abstract class `ExternalFitnessCalculator` was added for consistency and to suspend the algorithm as long as the population is not complete. Fig. 4.13 shows the class hierarchy.

4.2.2.f PopulationSize

`ConstantPopulationSize` restricts the size of the *best generation* to a constant value. Having multiple criteria e.g. stability and band-gap it might be useful to allow mutation and crossover on all stable alleles and therefore let the *best generation* grow. This can be achieved by using `GrowingPopulationSize`. (See Fig. 4.14)

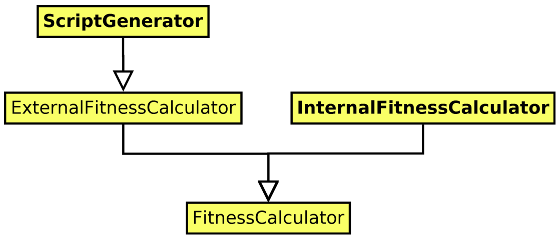


Figure 4.13: The `InternalFitnessCalculator` is used to calculate the fitness during the execution of the GA. This means the fitness of the offspring is calculated in a consecutive manner. If `ScriptGenerator` is chosen then a (python) script is generated which is sent to a cluster.

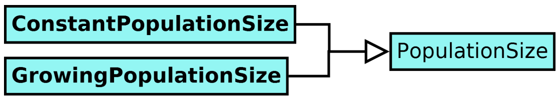


Figure 4.14: Having multiple criteria e.g. stability and band-gap one might want to keep all stable alleles as parents for mutation and crossovers - otherwise constant population sizes are chosen.

4.2.2.g Factory

`Factory` implements the factory pattern method[75]. Factories are used to create objects like alleles or populations. The reason not to create the instances directly is to assure that alleles/populations are correctly initialized (with the algorithm id, how they were created, et cetera) even if they were customized. (See Fig. 4.15)

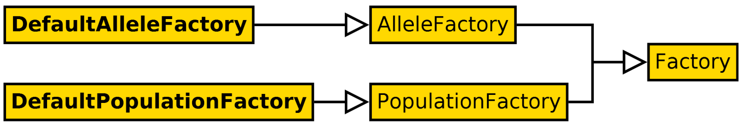


Figure 4.15: A factory is used to create instances of alleles and populations.

4.2.2.h Hook

Hooks can be used to monitor or access the state and for example print the most recent populations. Hooks are provided before and after loading the current state, the creation of the initial or any consecutive generation, replacement of failed alleles and the determination of the fitness of the alleles. Besides printing they are useful for debugging purposes. An example is given in section 4.2.5. `Visitor` `Visitor` uses the visitor pattern[75] to visits the state, all generations and alleles. It is used to create statistics (`Statistics`) create text (`TxtPrinter`) or HTML (`HTMLPrinter`) output. (See Fig. 4.16)

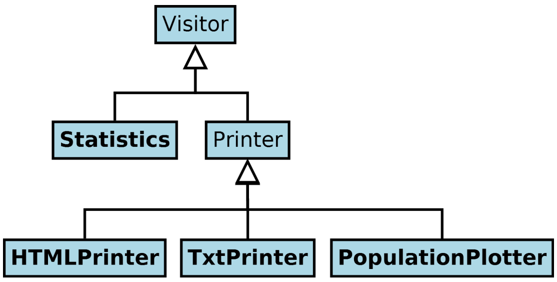


Figure 4.16: Visitors are used to create statistics, print and plot the algorithm's result. All processes that need to investigate *all alleles* or *all populations* use a visitor.

4.2.2.i ConvergenceControl

Different criteria can be chosen to stop the algorithm: `GenerationRepetitionConvergenceControl` halts the execution when the *best generation* doesn't change any more for n generations or `AvgFitnessConvergenceControl` stops when the average fitness of the *best generation* remains in a certain range. (See Fig. 4.17)

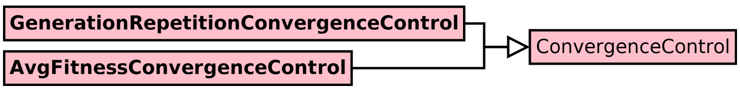

GenerationRepetitionConvergenceControl
AvgFitnessConvergenceControl
ConvergenceControl

Figure 4.17: The algorithm can be stopped based on the average fitness or when no new offspring are found any more.

4.2.2.j GenerationCreator

With problems using DFT calculations it is quite common that the fitness calculation fails or that the allele is vetoed. In some situations it is acceptable to just ignore these individuals, in others the calculations have to be fixed and resubmitted or simply replaced with a different one - the user has to choose the appropriate behavior. To support the automatic replacement of alleles, all classes that inherit from `GenerationCreator` implement a method that replaces the allele with a compatible one. This means if the failed allele was be created by a crossover, the replacement will as well created by a crossover. How to enable or disable replacing failed alleles is demonstrated later in the example discussed in section 4.2.5.

4.2.2.k The Refined Algorithm

The important components of the refined algorithm have now been introduced and we can follow now the execution of the algorithm as shown in Figure 4.18. The labels of the processes (the green boxes) denote the names of the involved classes.

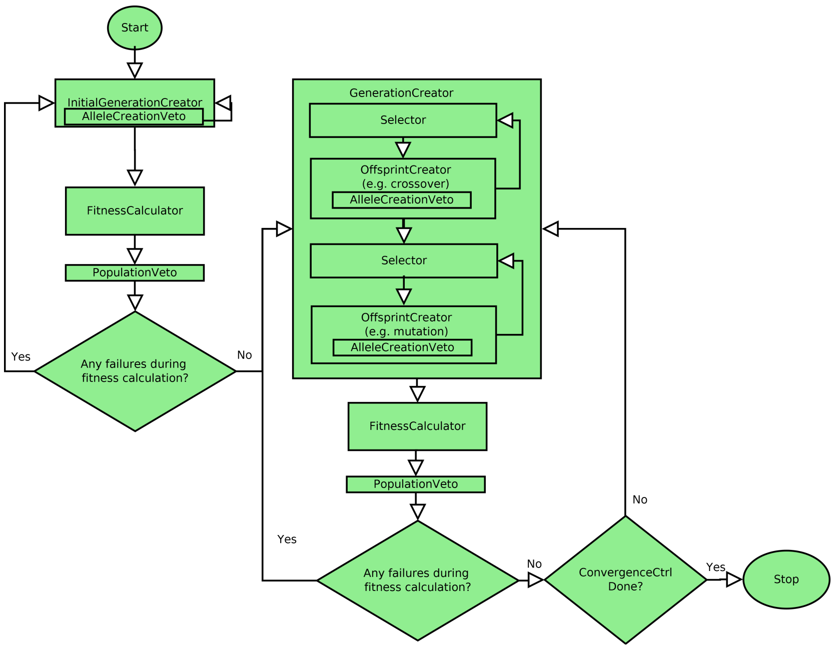


Figure 4.18: A detailed flow diagram of the implementation of the GA. The identifiers in the boxes denote the name of the involved (abstract) classes. This makes it easier to connect the implementation with this diagram.

While stepping through the algorithm the involved abstract class instances are explicitly mentioned. When considered helpful a possible concrete class is used to make the example easier to read. In this case the notation `abstract/concrete class` is used.

First an initial population is created by the `InitialGenerationCreator` instance. Each of the created alleles is then checked to see if it is vetoed by the `AlleleCreationVeto/DefaultAlleleCreationVeto` instance to assure that there are no duplicates. Vetoed alleles are replaced right away. Then the fitness calculation of the initial generation is started by the `FitnessCalculator` instance. If there are failed alleles and it was chosen to replace them, the loop starts again - but this time the `InitialGenerationCreator` only replaces the failed alleles. Fixing the failed alleles and calculation of the fitness is repeated until the population is complete. Then the creation of the next generation starts: the `GenerationCreator/DefaultGenerationCreator` instance creates mutations and crossovers. We assume that the mutation class instance is of type `vectors.MutationX`, and the crossover `vectors.Crossover2`. We start with the crossovers: the `Selector` chooses two parents from the current *best generation*, crosses them and creates the offspring and checks if the `AlleleCreationVeto/DefaultAlleleCreationVeto` accepts it. The same is performed for the `vectors.MutationX`. When the number of offspring has reached the population size, the fitness is calculated. The next steps are to check for duplicates with the `DefaultPopulationVeto/PopulationVeto`, checked for failed calculations and replace them, if the user requested to replace the failed alleles. Once the generation is complete the `ConvergenceCtrl/GenerationRepetitionConvergenceControl` stops the algorithm if there is no improvement for three generations, otherwise another cycle of crossovers and mutations is started.

One simplification was made in the flow diagram: When the fitness is calculated externally on a computer cluster, the algorithm exits. After restart the new fitnesses of the alleles are updated and the algorithm continues where it stopped.

4.2.3 GA and CMR

The GA can profit from a combination with CMR in many ways: the GA gets database access to its results, can add keywords, scripts, other calculated properties. CMR also allows to store and group arbitrary data and makes it therefore possible to serialize the internal state of the GA as well. Storing the internal state allows the algorithm to stop while the fitness of the alleles is externally calculated (on a computer cluster for example). Upon restart, the algorithm loads the state, updates the alleles with the new results and then continues where it stopped. Since the state is stored with CMR as db-files and can be submitted to the CMR database, the PHP/HTML interface can be used to visualize a run as shown earlier in Fig. 2.17 and in Fig. 2.15.

The **state** of the algorithm is defined by a name of the run and it's history, which is all ever calculated alleles. This is enough to determine the *best generation* that is needed to create the next generation - but not sufficient for the PHP/HTML interface that does not know how to determine a *best generation*. Therefore the *generations* and *best generation* are serialized explicitly. The state of the GA is stored hierarchically as illustrated in Fig. 4.19: *state* has references to *generations* which maintains a list of previous *generations* and to *best generations* that maintains a list of *best generations*. The same applies the other boxes e.g. *generation 1* (the initial population) has a reference to the alleles 1-4 and the *best generation 2* (the best alleles up to and including the second generation) references the alleles 2,3,6 and 7. Every box is a CMR group. As we remember groups contain fields and references to their group members. The arrows visualize the references to the group members which may themselves have members.

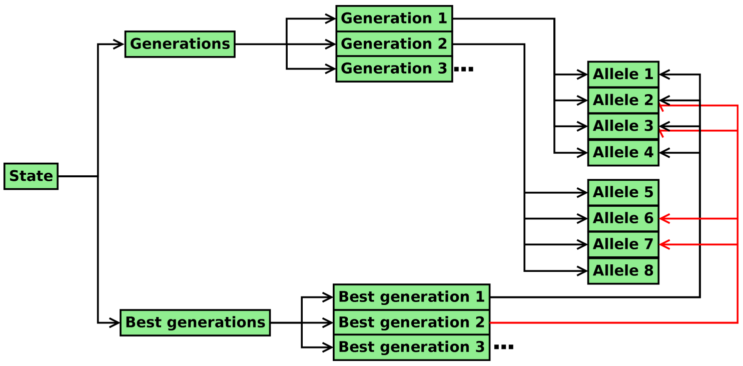


Figure 4.19: The data structure of the GA. Every box is a CMR group. The lines are references (links): e.g. **State** is a group and *references* the groups **Generations** and **Best generations**. **Generations** contains **Generation 1**, **Generation 2**, **Generation 3**. Every allele is a group member of exactly one *generation*, but can be member of multiple *best generations*(Fig. 4.6) if it has a good fitness value. For example **Allele 2** was created in the first generations and is among the best alleles in the **best generation 1** and **best generation 2**

The *state*, *generations*, *best generations*, *generation n* and *best generation n* store only references to their members. The *allele n* however are more interesting: they contain all user defined variables, the state of the allele (successful, failed, vetoed, submitted), the name of the parents, the way they were created (mutation, crossover), the before and after hash, the name of the output file and most important: the fitness. During the determination of the fitness the state of the allele is constantly updated - later it is only accessed for statistical purposes or when printing.

The classes that are involved in the serialization of the state are shown in

Fig. 4.20. All objects that store data in db-files inherit from `CMRObject`. The most prominent is the `CMRAllele` that contains information about an allele and the custom alleles inherit from. `CMRDirState` stores the state on disk and is able to send a copy to the CMR database as well. The advantage of storing the state on disk is that all data is immediately available, a submission to the database takes always a bit longer and the time depends on how heavy the CMR database is used. Finally `GenericCMRObject` enables storing of data as db-file directly and is used internally for example to write the list of the *generations* and *best generations*.

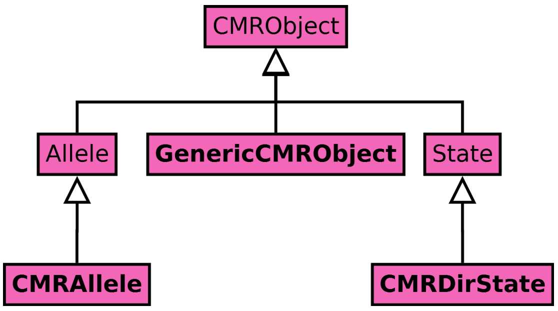


Figure 4.20: The classes involved in the serialization of the state. All classes that store state or state related information inherit from `CMRObject`. `CMRDirState` stores the state on the disk and optionally sends a copy to the CMR database. The `GenericCMRObject` enables storing any kind of data to a db-file; it is used for example for writing the list of *generations* and *best generations*. `CMRAllele` serializes itself upon changes (e.g. updated fitness) and is used as a base class for customized allele implementations.

Suggested improvement
When the total number of alleles starts to become large the check for duplicates begins to be expensive because all alleles are loaded to memory. A solution to this problem is to use a CMR query to retrieve a list of all *before* and *after hashes* and then compare the new offspring's hashes with the items in the list.

4.2.4 Example 1: Prediction of Stable Electro-catalytic Nanoparticles with the Genetic Algorithm

Energy conversion processes as they are used in batteries depend on efficient catalysts. Often a substrate on a bulk material is used, but it has been shown that under certain conditions nanoparticles are more reactive[76, 77]. A possible reason is that they expose more or different sites which can accelerate the catalytic process. The structure and sites of the nanoparticle are influenced by its composition and is neither easy to predict nor can it be reliably extrapolated. Another problem is that nanoparticles are inherently metastable especially under reaction conditions with high pressure and temperature as used in ammonia production[78, 79]. For this reason different methods are used to predict the structure and stability in a first step and in a second the reactivity. Steen Lysgaard a PhD student at CAMD at the time of writing conducts a study with the goal to predict stable electro-catalytic nanoparticles for reactive conditions for ammonia, methanol and higher alcohol production from N_2 and CO_2 respectively. Because there is no known gradient method to find these catalysts the Genetic Algorithm (GA) is used. Fig. 2.17 shows the couple of generations of a run of the GA with a copper and nickel nano-particle that could be efficient for CO_2 fixation. In this case the GA starts with a population of m randomly initialized unit cells with copper and nickel atoms. The fitness value is the total energy and the mutation operation moves the atom positions a tiny distance in a random direction. The crossover operates on two alleles, cuts them in halves and combines them with the halves of the other allele. This example uses the `AlleleAcceptVeto` to detect duplicates after the calculation of the fitness. (The relaxation moves the atoms to physically more favorable positions, therefore there is a high chance to get duplicates.) The project is still in its starting phase therefore no conclusions or more details about actual runs can be presented yet.

4.2.5 Example 2: Function Minimization

This example shows how the classes are combined to get a running GA that minimizes a function $f(x, y)$. To make it more interesting we calculate the fitness in script and submit it to a computer cluster for evaluation. The only components that need to be implemented is a template script that calculates the fitness, a script that defines the components, a custom allele and a class to create the initial alleles. The full code is available in the example directory of the implementation.

The class `MyAllele` (Fig. 4.21) is derived from `CMRAllele` and allocates the field `coord` which stores the coordinate. As seen before there are two hash values used to detect duplicates: one just after the allele is created and the other after the fitness was calculated. In our case the coordinates don't change with the calculation therefore the *before* and *after* hash defined in the constructor `__init__` are the same.

`get_name` returns a human readable expression that is used when printing intermediate or final results.

```
from ga.base.cmr_allele import CMRAllele

class MyAllele(CMRAllele):
    def __init__(self):
        hash_def = {"before_hash":["coord"], "after_hash":["coord"]}
        CMRAllele.__init__(self, hash_def, 4)


    def get_name(self):
        """returns the name of this allele."""
        res = "%.5e_%.5e" % (self["coord"][0], self["coord"][1])
        return res.replace("+", "").replace("-", "")
```

Figure 4.21: `MyAllele.py`

To create the first generation we need alleles initialized with random coordinates. These are provided by the class `MyInitialAlleleCreator` (Fig. 4.22) which is derived from `OffspringCreator`. The member function `create` returns a new allele. Every allele knows which class created it. The function `get_description` returns a string that describes the creator which is just the name of the class.

```
import random
from ga.base.offspring_creator import OffspringCreator

class MyInitialAlleleCreator(OffspringCreator):

    def create(self, generation_number=0):
        allele = self.new_allele(0, [])

        allele["coord"] = [random.randint(-900, 400), \
                           random.randint(-300, 800)]
        return [allele]

    def get_description(self):
        return "MyInitialAlleleCreator"
```

Figure 4.22: `MyInitialAlleleCreator.py`

The script template needs to implement the method `run` (shown in Fig. 4.23) that calculates the fitness and updates the fitness and the state of the allele. The allele variables are accessible with the dictionary `parameters`. Updating is done by calling `write_spreadsheet` which will write the result to a db-file and updates the allele as well. The full script with more explanations is available in the appendix F.4.

```
...
def run():
    from ga_examples.fe_ext_calc.functions import function1

    x = parameters["coord"][0]
    y = parameters["coord"][1]

    # calculate the function's result:
    result = function1(x, y)

    cmr_params = dict(parameters.items())
    cmr_params["fitness"] = result

    write_spreadsheet(cmr_params)
...
```

Figure 4.23: An extract from `script_template.py`. The full script is available in the appendix F.4.

The only thing left is to build the algorithm. This file is generally called `main.py`. We will now step through it. The variable and function names are generally of intuitive nature and reflect the usage. First we define the imports:

```
import os

from ga.base.runtime import Runtime
...
```

Next the runtime instance that stores settings and the **name** of the run are defined. It is common to run a genetic algorithm multiple times to see if it converges to the same result. Therefore is is important to use a different name for every run.

```
setup = Runtime()
setup["name"] = "function_example_run_1"
```

Then we define the location where to store the state, the generations, alleles and the results of the calculations, ...

```
# the directory where all genetic algorithm results are stored
genetic_algo_dir = os.path.join(os.environ[ENV_HOME], "ga_runs")
#the name of this file
setup["main_file"] = "main.py"
#directory where the state is stored
setup["state_dir"] = os.path.join(genetic_algo_dir, setup["name"])
setup["output_dir"] = os.path.join(setup["state_dir"], "output")
setup["calc_dir"] = os.path.join(setup["state_dir"], "calcs")
```

```
if not os.path.exists(setup["output_dir"]):
    os.makedirs(setup["output_dir"])
```

... the population size and that we look for the minimum (and not for a maximum) fitness value, ...

```
setup["population_size"] = 6
setup["higher_fitness_is_better"] = False
```

... that we request a constant size of the *best generations* ...

```
setup["population_size_control_instance"] = ConstantPopulationSize()
```

... and then set the file name for the log file and create an instance. The arguments in the constructor are all optional and have the following meaning: `output` sets whether the log messages are printed to the screen while running, the `file_handle` points to the log file, `store` sets whether to store the log file and `prefix` is a string prepended to every log message:

```
setup["log_file_name"] = os.path.join(setup["output_dir"], "log.txt")
log = open(setup["log_file_name"], "a")
setup["log_instance"] = Log(output=False,
                            file_handle=log,
                            store=False,
                            prefix="GA")
```

The convergence control regulates when to stop the algorithm. In this case we stop, if the top five alleles are the same for three consecutive generations:

```
setup["convergence_control_instance"]
          = GenerationRepetitionConvergenceControl()
setup["conv_ctrl_num_generations"]=3
setup["conv_ctrl_num_alleles"]=5
```

The state shall be stored on disk and with `db` set to `True` a copy is sent to the CMR database as well ...

```
setup["state_instance"] = CMRDirState()
setup["db"] = True
```

... and the fitness shall be calculated by the `ScriptGenerator`. `submit_-instance` defines that the script must submitted to the queuing system (`qsub`). Then there are two templates: the python template that was already shown and the shell script template that is used to define the environment variables before the created script will be run.

```
setup["fitness_calculator_instance"] = ScriptGenerator()

setup["submit_instance"] = Submit(["qsub", "-q", "small", "-l",
                                   "nodes=1:ppn=1", "-me"])

setup["py_script_template"] = "script_template.py"
setup["sh_script_template"] = "env.sh"
```

An example of the shell script template is given in Fig. 4.24. `#<FILL_START>` and `#<FILL_END>` are going to be replaced with the call to the python script when the template is applied.

```
#!/bin/sh

echo "Start time: `date`"
#<FILL_START>
#<FILL_END>
echo "End time: `date`"
```

Figure 4.24: A minimal shell script template `env.sh` that defines the environment variables that should be used on the computer cluster.

We want to use the default algorithm, ...

```
setup["ga_instance"] = GA()
```

... default factories for the population, alleles and vetoes, but our own allele `MyAllele` ...

```
setup["population_factory_instance"] = DefaultPopulationFactory()

setup["population_veto_instance"] = DefaultPopulationVeto()
setup["allele_creation_veto_instance"] = DefaultAlleleCreationVeto()
setup["allele_accept_veto_instance"] = DefaultAlleleAcceptVeto()

setup["allele_factory_instance"] = AlleleFactory(MyAllele)
```

... a ranking selector with a selection pressure of 1.5 (range is 1-2) ...

```
setup["selection_instance"] = RankSelector(1.5)
```

... create the initial population using the just defined initial allele creator ...

```
setup["initial_allele_creator_instance"] = MyInitialAlleleCreator()
```

... define a crossover of three alleles (triangulation), three mutations that modify the coordinate randomly in the ranges $[-200, 200]$, $[-10, 10]$ and $[-.1, .1]$. `offspring_creator_instance_ratio` defines how often the crossovers and mutations should be applied and `max_try_offspring_creation` defines that it should be tried 20 times to create an offspring before the algorithm is stopped. The reason for the limit is that if the population is not diverse any more it can happen that all crossovers result in already calculated alleles and the algorithm cannot continue any more.

```
setup["vector_name"] = "coord"
# EITHER
# setup["generation_creator_instance"]=DefaultGenerationCreator()
# setup["crossover_instance"]=CrossoverX(3)
# setup["mutation_instance"]=MutationX(-10, 10)
# setup["mutation_crossover_ration"]=(1, 1)
# OR
setup["generation_creator_instance"] = GeneralGenerationCreator()
setup["offspring_creator_instances"] = [CrossoverX(3),
                                         MutationX(-200, 200),
                                         MutationX(-10, 10),
                                         MutationX(-.1, .1)]
setup["offspring_creator_instance_ratio"] = [1,1,1,1]
setup["max_try_offspring_creation"] = 20
```

Now we define whether alleles should be replaced if they fail. This is chosen separately for the first and for all the following alleles.

```
# -1: stop on failed alleles
#  0: ignore failed alleles and continue
#  1: replace the failed alleles
setup["replace_first_gen_failed_alleles"] = 1
setup["replace_failed_alleles"] = 1
```

During the execution summary containing some statistical information are written as txt and HTML files:

```
setup["txt_printer_instance"] = txt_printer = TxtPrinter()
setup["html_printer_instance"] = html_printer = HTMLPrinter()

setup["statistics_instance"] = Statistics()
```

Then the hook instance and the message that should appear when exiting is defined followed by the registration of the `on_exit` hooks.

```
# register, when to print:
setup["hook_instance"] = hook = Hook()

def end_message():
    print "Locations:"
    print "Main file                       - %s" % setup["main_file"]
    print "Log file                        - %s" % setup["log_file_name"]
    print "State directory                 - %s" % setup["state_dir"]
    print "File repository directory:      - %s" % setup["calc_dir"]
    print "Output directory:               - %s" % setup["output_dir"]
    print "All generations:                - %s" % txt_output
    print "Best for every generations:     - %s" % txt_best_output
    print "All generations:                - %s" % html_output
    print "Best for every generations:     - %s" % html_best_output
    print "Log file                        - %s" % setup["log_file_name"]

html_output = os.path.join(setup["output_dir"],
                           "generations.html")
html_best_output = os.path.join(setup["output_dir"],
                                "best_generations.html")

hook.register("on_exit",
              html_printer.write_generations,
              {"file":html_output})
hook.register("on_exit",
              html_printer.write_best_generations,
              {"file":html_best_output})
hook.register("on_exit",
              end_message)
```

Finally everything is defined an the algorithm can be started.

```
run(setup)
```

When `main.py` is called the first time it will create the initial population and send the scripts to calculate the fitness to the computer cluster. Any subsequent execution will load the state and if all fitnesses are available continue.

To get started with a customized GA it is best to use one of the examples that comes along with the source code. The alternative is to create a minimal script Fig. 4.25 and follow the instructions.

```
import os

from cmr.static import ENV_HOME
from ga.base.runtime import Runtime
from ga.base.run import run

genetic_algo_dir = os.path.join(os.environ[ENV_HOME], "ga_runs")

setup = Runtime()
setup["name"] = "fe_ext_calc2"
setup["state_dir"] = os.path.join(genetic_algo_dir, setup["name"])

run(setup)
```

Output:

```
Exception: Undefined items found:
Undefined instance found: 'ga_instance'.
Undefined instance found: 'hook_instance'.
Undefined instance found: 'state_instance'.
Undefined instance found: 'convergence_control_instance'.
```

Figure 4.25: A minimal `main.py` file. The selected components are are evaluated during execution and the missing ones will be pointed out.

4.2.6 Discussion

The reason for us to use Python is that it supports object oriented design, is easy to learn, available for free, does not need an external compiler and Python programs are distributed with the source which makes it accessible to all users. There are not so many GA frameworks available in Python. There is only one paper[80] that provides basic hints about how to get started, other references note only that they used Python, but don't provide the framework. A search on the Internet reveals a couple of generic GA implementations. First: **Pyevolve 0.5**[81] is the best documented one and offers a modular design, nice statistics plots, a few selection, mutation and crossovers methods, is easy to install and is available with a license compatible to GPL. The drawback of this framework is that it is tricky to specify more than one mutation or crossover type and that the determination of the fitnesses of the alleles cannot be executed in parallel. An adaption of this code to work with CMR would have resulted in a considerable amount of recoding. Second: **Distributed Evolutionary Algorithms in Python (DEAP)**DEAP[82] is available under the GNU Lesser GPL, and supports parallel execution since version 0.7 as well. The approach of DEAP is similar to our framework: the algorithm is first set up by choosing pre-made components and then it is executed. The project was started at approximately the same time as the presented implementation. If it had been available earlier we might have used that project as a base for the present implementation of the GA with CMR.

We have presented a very flexible implementation of the genetic algorithm consisting of many pre-made components that can be combined to build a customized genetic algorithm. Advantages over other frameworks are traceability with hooks, full control over the algorithm, vetoing during the creation of the new alleles and vetoing after the fitness is determined. Furthermore the calculation of the fitness for a whole generation can be performed in parallel and special customized crossover or selection operators can easily be added. The tight coupling with CMR is an advantage when the results are added to the CMR database and the calculation of the fitness is expensive. The framework is however not suitable for a problem with a simple fitness function, thousands of generations and many runs because it cannot compete with an implementation that was coded in a compiled language as c++ for example.

4.3 DFT based screening of ternary alkali-transition metal borohydrides - a computational materials design project

The development of ecological energy carrier that produce little or no CO_2 emission is interesting for the transportation sector and other industries storing or converting energy. In order to be competitive with fossil fuels such a carrier must be thermodynamically stable, efficiently convertible and comparably safe. Hydrogen being lightweight and able to be generated from renewable energy sources is a good candidate, but occupies roughly 11 liters of volume per gram. Instead of storing it in pure form, metal borohydrides[83] have been proposed for reversible chemical storage. The alkali based binary metal borohydrides (e.g. $LiBH_4$) turn out to be too thermodynamically stable[84, 85, 86] while the alkaline earth based ones are kinetically too slow[87] and the transition metal ones are either unstable or hardly reversible[88]. The hope is that more complex alloy of these different borohydrides might provide better results.

The performed screening on ternary metal borohydrides for reversible hydrogen storage investigates the stability and decomposition of compounds in the form of 1 alkali metal atom (Li, Na, K) and an alkaline earth or 3d/4d transition metal atom with two to five $(BH_4)^-$ groups with different structures (trigonal, tetrahedral, octahedral or free coordination). Most of the calculations were done during the CAMD Summer School 2008 on density functional theory as a materials design project where over 100 PhD student and postdocs contributed. Only a few calculations were subsequently performed based on the gained insight. In order to get useful comparable results, all calculations were performed with unified parameters where possible: we used the software package Dacapo with the RPBE exchange-correlation functional[89] and the same values for the cutoffs (energy and density grid), k-points for all the calculations and the all relaxations were performed with a quasi-Newton method[90]. Dacapo (which is based on DFT) was chosen because DFT allows to estimate the result for complex borohydrides based on simple model structures[91].

Possible candidates for reversible hydrogen storage were identified with respect to the stability against phase separation ($\Delta E_{\mathrm{alloy}}$) and the decomposition pathway ($\Delta E_{\mathrm{decomp}}$).

For $LiSc(BH_4)_4$ the stability against phase separation looks as follows:

$$\Delta E_{\mathrm{alloy}} = E_{\mathrm{LiSc(BH_4)_4}} - \left(E_{\mathrm{LiBH_4}} + E_{\mathrm{Sc(BH_4)_3}}\right). \quad (4.3)$$

Since most of the true decomposition pathways are unknown or if known, very different from each other, one was chosen that decomposes the alloy into an alkali – or alakli earth hydride, the transition metals, the boron atoms and H_2. For $LiSc(BH_4)_4$ this looks as follows:

$$\Delta E_{\mathrm{decomp}} = E_{\mathrm{LiMn(BH_4)_4}} - (E_{\mathrm{LiH}} + E_{\mathrm{Mn}} + 4E_{\mathrm{B}} + 7.5E_{\mathrm{H_2}}). \tag{4.4}$$

Values between -0.5 and 0 were considered interesting for $\Delta E_{\mathrm{decomp}} < 0$ and for $\Delta E_{\mathrm{alloy}}$ values smaller than 0. The reference energies for E_{LiH}, E_{Mn}, E_{B} and $E_{\mathrm{H_2}}$ were calculated beforehand. From a total of over 700 structures and over 5000 calculations, we found 22 potentially stable alloys with good decomposition energies shown in Fig. 4.26 and Fig. 4.27.

A few of the identified candidates have been synthesized before and confirm the trends: this are $\mathrm{LiFe(BH)_3}$,[92] $\mathrm{LiAl(BH_4)_4}$,[93] $\mathrm{(Li/Na)Mn(BH)_{3,4}}$[94] and $\mathrm{(Li/Na)Zn(BH_4)_3}$,[95].

The components that made this screening possible are: soft and hardware infrastructure, templates for the calculations with initial guesses for the positions, and the collection and analysis of the results. In the following we will discuss the parts that provided us with insight for the later development of CMR. The details about the template creation and the numeric parameters can be found in paper C.

4.3.1 Requirements, Infrastructure and Organization

The requirements for the presented framework were that the participating researchers must be able to work and submit their resulting data files concurrently to a repository. The results should be stored in a traceable and modifiable way and the design should prevent technical problems like accidental overwriting of already present results. The researchers would be divided into 32 groups and each group would investigate two borohydride alloys each in 8 different structures, for example $\mathrm{LiSc(BH_4)_3}$ *and* 4 in trigonal, tetrahedral, tetra/octahedral (Li in tetra, Sc in octahedral coordination) and octa/tetrahedral structure. Each structure relaxed the hydrogene position and the unit cell volume. The best structures were then relaxed with no fixed atoms which resulted in a slightly distorted structure that we called “other”. The submitted results should then be displayed on a web page after checking their validity. No new infrastructure should be set up, instead the Niflheim[96] cluster, a file server, a web server, python, a

	wt.% [kg H_2/kg material]	ΔE_{alloy} [eV/f.u.]	ΔE_{decomp} [eV/H_2]
$LiNa(BH_4)_2$	13.5	-0.020	-0.581
$KZn(BH_4)_3$	8.1	-0.349	-0.423
$KAl(BH_4)_4$	12.9	-0.138	-0.416
$NaAl(BH_4)_4$	14.7	-0.279	-0.373
$KCd(BH_4)_3$	6.2	-0.005	-0.352
$NaZn(BH_4)_3$	9.1	-0.358	-0.344
$LiAl(BH_4)_4$	17.3	-0.391	-0.311
$KFe(BH_4)_3$	8.7	-0.116	-0.282
$LiZn(BH_4)_3$	10.4	-0.362	-0.243
$NaFe(BH_4)_3$	9.8	-0.141	-0.206
$KMn(BH_4)_4$	10.5	-0.148	-0.174
$NaNb(BH_4)_4$	9.2	-0.128	-0.165
$KCo(BH_4)_3$	8.5	-0.089	-0.161
$NaMn(BH_4)_4$	11.7	-0.284	-0.131
$KNi(BH_4)_3$	8.5	-0.120	-0.116
$LiFe(BH_4)_3$	11.3	-0.141	-0.104
$LiNb(BH_4)_4$	10.1	-0.194	-0.097
$NaCo(BH_4)_3$	9.6	-0.143	-0.090
$KRh(BH_4)_4$	8.0	-0.058	-0.079
$LiMn(BH_4)_4$	13.3	-0.358	-0.063
$NaNi(BH_4)_3$	9.6	-0.164	-0.043
$NaRh(BH_4)_4$	8.7	-0.033	-0.016

Figure 4.26: Candidate structures for reversible hydrogen storage with alloying energies $\Delta E_{\text{alloy}} < 0.0$ eV/f.u. (formula unit) and decomposition energies $\Delta E_{\text{decomp}} < 0.0$ eV/H_2.

programming language, subversion[97], a file versioning tool and cron[98], a job scheduler should be combined in a workflow to be able to meet the requirements.

4.3.2 Workflow

This paragraph describes how the given infrastructure components were joined to enable the collection of the data and later the analysis. An illustration showing the organization and the flow of the data is shown in Fig. 4.28.

In order to avoid possible naming conflicts and enable traceability, every group

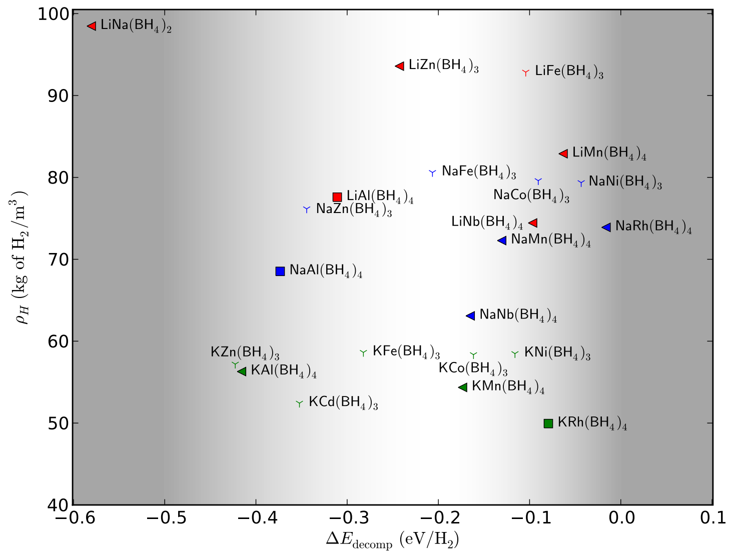


Figure 4.27: The hydrogen density (kg H$_2$ m^{-3}) as a function of the decomposition energy for the 22 alloys with $\Delta E_{\text{alloy}} \leq 0.0$ eV/f.u. and $\Delta E_{\text{decomp}} \leq 0.0$ eV/H$_2$; Colors: Li (red), Na (blue) and K (green).
local coordination: tetra (□), octa (○), octa-tetra (△), tetra-octa (+), tetra-tri (⋎), other (◁) .

got a name and a directory in the **subversion repository** assigned (camd001, camd002, ...). For every structure that the group members calculated they created a subdirectory that contained the script. Subversion assigns a (unique) revision number for every change. For now we assume that we added the subdirectory `LiMnB4_octa` and subversion assigned the revision number 505. The output files from Dacapo were then as a consequence named `LiMnB4_octa.nc.505`. If the the script was modified later it would get a new revision number and so would the output files. This guaranteed that the results remained traceable and prevented accidental overwriting. When satisfied with the result file the researchers would move the Dacapo output file to the **calculation repository**. (The Dacapo output file were not added to the subversion repository because of their size, which can easily exceed 40 MB and the fact that versioning is not needed, because they will never be modified.)

To make efficient searches possible all needed information was extracted from the Dacapo output file, enhanced with a few calculated properties and written as a python pickle file. Python pickle files are used in the python programming

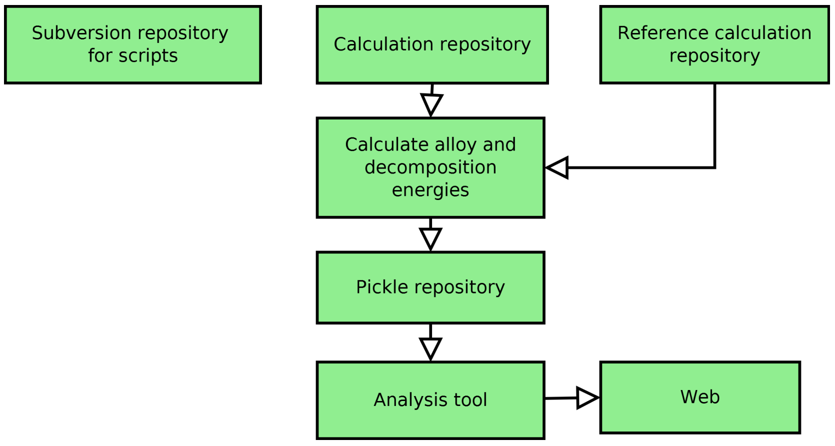

Subversion repository for scripts
Calculation repository
Reference calculation repository
Calculate alloy and decomposition energies
Pickle repository
Analysis tool
Web

Figure 4.28: Organization and flow of the data. The scripts for the calculations are stored in the subversion repository while the Dacapo output files are moved to the calculation repository. For efficiency reason the interesting data is extracted from the Dacapo output files, new fields like the weight percent of H, the decomposition and alloy energy are calculated with the help of the reference energies and then written to the pickle repository. The analysis framework then reads the pickle files, applies the search/selection criteria to create plots and files for a static web page which is then accessible for the participants on the Internet.

language to write objects like dictionaries to a file. This avoided parsing the Dacapo file and recalculating the properties for every run. These files were written to **pickle repository**. The information stored in the pickle files were the name of the structure, the formula, the weight percent of H, the decomposition and alloy energies and a few parameters of the calculation. The **analysis tool** allowed to write queries that search the pickle repository. For example it is possible to select all results that contain lithium and display their decomposition energies.

4.3.3 Analysis Tool

To analyze, plot and show the data on the Internet a simple python analysis tool was built that runs every few minutes and updates the current results. When run, the tool would first load the data from the pickle repository into memory and then execute the queries to select the requested data. The queries are performed in a very similar way as with CMR that is described in the section 2.4. To create the data in Fig. 4.27 for example, all results with $\Delta E_{\mathrm{alloy}} \leq 0.0$ eV/f.u. and $\Delta E_{\mathrm{decomp}} \leq 0.0$ eV/H_2 would be selected and then passed on to a specially tailored module that creates the plot and assigns the correct symbols and colors. All plots were created with matplotlib[99], a graphics tool available for python. The content of the static web pages was selected in the same way, but then written as HTML.

4.3.4 CAMD Summer School Conclusion

The CAMD Summer School project was a big success: we found about 20 potentially stable alloys with good decomposition energies and were able to collect results performed by many researchers concurrently, could trace and correct errors and create a system to show and analyze the data almost just-in-time on the Internet. This project laid the foundation for the future development of CMR and inspired us especially in the way the analysis was performed and data was collected.

Chapter 5

Summary and Outlook

We have created CMR, a modular and extensible framework to aggregate, store, monitor, analyze, present and share data from electronic structure calculations. The framework is scalable and can be deployed as a single user system without any database software or as multi user system to serve a whole institute (requires database software). The user interfaces satisfy the needs of casual users as well as power users: CMR provides an intuitive web interface and a programmable interface. The problem of storing results from different codes is solved by converting all data to a newly developed file format, the db-files. Db-files have the advantage that the content can be validated and the data are accessible without third-party software. To make analysis and finding results easier, they can be attributed with custom keywords and variables. The concept of (calculation) groups enables data to keep their natural connection also in the database. A group could for example contain all calculations that were needed to determine a chemisorption energy or all calculations belonging to a project. Agents, autonomous background processes are used to create tables for the user interfaces, but can also be assigned to perform automatic data analysis.

The data collection part of CMR is implemented in Python, and therefore some of the processes would run faster, if a different implementation language had been chosen. This is noticed when parsing or validating many files, however we think that the advantage of having an easy to learn powerful implementation language is a big advantage in terms of maintainability and ease of use. We show with the presented case studies that our approach is versatile and that it works. CMR is deployed at CAMd in two instances: one is the internal database that can used by all researchers, and the other is the public database where we put high-quality data of published articles.

Based on the current CMR infrastructure the next layer of abstraction can be built: For typical tasks such as calculating reaction, atomization or adsorption energies, calculation templates should be created. They reduce user errors and by using standardized keywords and field names makes it easier to trigger automated categorization and data analysis. These templates are a first towards letting researchers focus on data-mining instead of writing scripts for calculations.

CMR is part of Quantum Materials Informatics Project[14], which aims at establishing the core technology for integrated computational materials design.

CMR is available under GPL from `http://wiki.fysik.dtu.dk/`.
The high-quality CMR database can accessed at `http://cmr.fysik.dtu.dk/`

Appendix A

The Computational Materials Repository

D. D. Landis, J. S. Hummelshøj, S. Nestorov, J. Greeley, M. Dułak, T. Bligaard, J. K. Nørskov and K. W. Jacobsen

Accepted by Comput. Sci. Eng., 2012

The Computational Materials Repository

D. D. Landis[1], J. S. Hummelshøj[2], S. Nestorov[3], J. Greeley[4], M. Dułak[1], T. Bligaard[2], J. K. Nørskov[2] and K. W. Jacobsen[1]

[1]Technical University Denmark, Center for Atomic-scale Materials Design, DK-2800 Lyngby, Denmark
[2]SLAC National Accelerator Laboratory, SUNCAT Center for Interface Science and Catalysis, Menlo Pk, CA 94025 USA
[3]University of Chicago, Department of Computer Science, Chicago, IL 60637 USA
[4]Argonne National Laboratory, Center for Nanoscale Materials, Argonne, IL 60439 USA

January 20, 2012

Abstract

The possibilities for designing new materials based on quantum physics calculations are rapidly growing, but this leads to a significant increase in the amount of computational data created in relation to such design efforts. The Computational Materials Repository (CMR) addresses this data challenge and provides a software infrastructure that supports the collection, storage, retrieval, analysis and sharing of data produced by a number of modern electronic-structure simulators.

Keywords: H.3.2.d Storage/repositories < H.3.2 Information Storage < H.3 Information Storage and Retrieval < H Information Technology and Systems,
H.3.3.b Information filtering < H.3.3 Information Search and Retrieval < H.3 Information Storage and Retrieval < H Information Technology and Systems,
J.2.i Physics < J.2 Physical Sciences and Engineering < J Computer Application

1 Introduction

The design of novel and versatile materials is an issue of central importance for society. This is exemplified by a significant current focus on discovering new materials for energy conversion and storage to provide a sustainable alternative to the fossil-based fuel economy. Atomic-scale calculations are becoming increasingly important in strengthening our ability to meet this challenge, as they over time have provided an

ever-improving alternative to expensive experiments. One common aspect of conducting computational atomic-scale materials design is the need for carrying out calculations on a large number of materials. This poses a challenge in terms of systematic storage of the calculations, enabling easy retrieval, comparison, and analysis. We present a software infrastructure named the Computational Materials Repository (CMR), which addresses this problem by implementing a modular framework in python, providing tools for collecting, storing, grouping, searching, retrieving, and analyzing data generated by a number of modern electronic-structure simulators. The focus is in particular on Density Functional Theory (DFT) which represents a favorable trade-off between speed and accuracy for the treatment of "few-hundred-atom" systems highly relevant for understanding physical and chemical properties of materials. CMR can be used for single user projects, but can also make user of a MySQL database[1] which enables inter-group collaborations and allows to process significantly more data.
CMR is currently in use in our groups, and we provide it under the open source GPL license to any group or individual who may find it useful.

2 The Computational Materials Repository

From experience with the Atomistic Simulation Environment (ASE)[2,3] and its interfaces to legacy as well as state-of-the-art electronic structure codes, we've learned that python[4] is well suited for our purpose; it is a high-level programming language used in research and industry due to its intuitive and powerful syntax, that fits the needs of newcomers as well as expert programmers. As with the ASE we would like users to be able to benefit from CMR at several levels of complexity. At the simplest level the CMR can be used by a single user with his/her data in an ordinary file system without installing a database. Power-users who want to access data faster and with more flexibility will install a MySQL database and use one or more of the CMR interfaces to benefit fully from the system.

The database system that we describe here is tailored to store, retrieve, and analyze data related to properties of matter at atomic scale. A fairly large number of so-called electronic structure codes exist today,[5] and it is the ambition of CMR to work broadly with many of these codes. The different codes can, however, have quite different output file formats, even though the output might contain relatively similar information. Our solution to the issue of different file formats is to initially convert the data to intermediate db-files (also called cmr-files). The variable names used in the db-file are whenever possible the same as in the original file format, which allows users to become quickly familiar with the use of the database. The reasons for using intermediate db-files are that they permit to verify contents , collect data without direct access to the database, and to exchange data with other parties. The CMR thereby introduces more flexibility in storing different types of calculations, new possibilities for intra- and inter-group collaboration, and better support for third party analysis tools than the stand-alone Java-based VMDF package.[6]

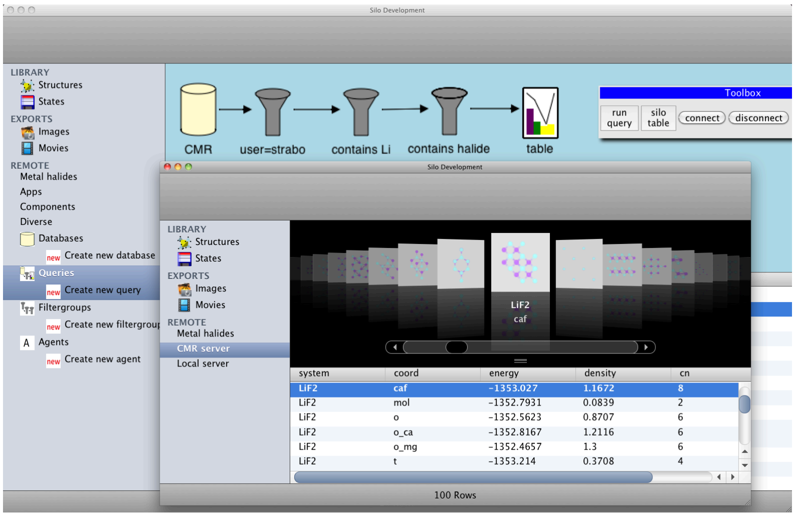
2.1 Accessing the database

Communication with a database is done by writing so called queries. For example `SELECT * FROM db.tbl` would return all data from a table named `tbl` that is contained in a database called `db`. This is not very user-friendly since queries tend to become long and complicated; moreover, queries depend on the structure of the underlying database; so when the database structure changes, previously written queries become invalid. CMR handles this burden of writing queries by providing three interfaces: a python, a PHP, and a HTML/JavaScript interface.

The most powerful interface is the **python interface**. It allows users to retrieve a subset of the data by selecting keywords, atomic numbers or ranges of data and then do further processing. This interface is flexible with respect to the location of the data and can query a database as well as a db-file repository. This assures that people starting with a db-file repository can easily switch to using a database later on. Using python enables the user to perform advanced operations like grouping results based on keywords and writing results back to the database.

PHP is a general-purpose scripting language that runs on a web-server. The **PHP interface** makes it possible to visit a CMR server with a web-browser and provides access to the database in a manner similar to a state of the art search-engine; searches are performed by entering keywords, atom names and data ranges for variables. Additionally atomic structures and all files (like calculation scripts and graphics) included in a db-file can be viewed, which turns out to be very useful, especially if the keywords are not accurate or it is not clear how the data were calculated. Results can be downloaded, further analyzed, grouped and re-uploaded with the python interface.

Although the python interface offers a lot of functionality, it is in many cases convenient to use a graphical user interface where for example atomic structures can be visualized as in the **HTML/JavaScript interface** `silo`. `Silo` runs in state-of-the-art browsers such as Chrome, Opera, Safari or Firefox (the local caching of data in a sqlite database is not supported in Firefox). **SiGUI** (Simple Graphical User Interface), a plug-in for `silo` enables users to create queries visually without the need to know a database query language. In Fig. 1 we show a query that retrieves data from a user named ”strabo”, selects results that contain the atom ”Li”, and finally picks the results that contain the keywords ”halide”. The results are presented in a cover flow showing the atomic structures above a table with the corresponding values.



system	coord	energy	density	cn
LiF2	caf	-1353.027	1.1672	8
LiF2	mol	-1352.7931	0.0839	2
LiF2	o	-1352.5623	0.8707	6
LiF2	o_ca	-1352.8167	1.2116	6
LiF2	o_mg	-1352.4657	1.3	6
LiF2	t	-1353.214	0.3708	4

100 Rows

Figure 1: The silo plugin `SiGUI` enables users to create queries in a graphical user interface.

2.2 Analyzing data

Analyzing the data is one of the most important aspects of CMR. Without analysis the database would just present an overwhelming amount of strings and numbers of little use . It is therefore important that the analysis be both thorough and flexible.

The first feature to use for analysis is the **taxonomy** which consists of system generated tags/keywords which describe the data. Some of these are quite obvious like the identity of an atom being “Nitrogen” and the numbers which describe the position of a given atom in space. These are the kinds of keywords which are already identified with the initial upload of data. However, there are also other keywords which can be automatically deduced. If a particular system contains a nitrogen atom surrounded by 3 hydrogen atoms at typical bond distances, this is an indication that the system contains an ammonia molecule. It may therefore be appropriate to tag the system with the keyword “ammonia” so that later searches can easily find the calculations involving this molecule. Such automatic assignments of keywords are generally complex and are therefore left to the users.

Another feature for analysis purposes is the use of **folksonomy**. This term from the internet lingo describes “a system of classification derived from the practices and methods of collaboratively creating and managing tags to annotate and categorize content”.[7] In other words, the database should take advantage of all the useful knowledge available from its users in classifying the content. One user might for example carry out studies related to how small molecules bind to a nickel surface and therefore decide to add a keyword “chemisorption” to a set of calculations. This is very useful later on for other users who might want to study the same subject and who with a simple search can identify relevant calculations, which have already been carried out.

The taxonomy and folksonomy tagging becomes even more dynamic by introducing **agents**. By an "agent" we mean a program or process running in the background (or in some cases executed by the user manually) to query the database using taxonomy and folksonomy classification to retrieve data, perform operations, group results, or create tables. An example of a simple agent could be a piece of code identifying ammonia molecules as described above. This code could run automatically every time new data are uploaded to the database and ensure that there would be an up-to-date list of ammonia molecules entering calculations in the database.

An agent which has the purpose of calculating chemisorption energies is a little more involved. The calculation of a chemisorption energy requires three electronic total energy calculations:

- The adsorbate in the gas-phase (X)
- The clean surface (Y)
- The surface with the adsorbate (i.e. a molecule bound to the surface) (Z)

The chemisorption energy, E_{chem}, is then calculated as $E_{chem} = E_Z - E_X - E_Y$. The agent first finds all solutions for X, Y, and Z that satisfy the following criteria:

- X.keywords *contains* adsorbate
- Y.keywords *contains* surface
- Z.keywords *contains* surface+adsorbate
- X.ads=Z.ads
- Y.surface=Z.surface

This list can be supplied with more conditions to for example make sure that the three calculations in question are compatible in terms of system size, convergence, and other parameters. Based on all the discovered triplets of calculations, the agent builds a new table in the database with the chemisorption information. This agent could run periodically and create a table which is accessible to all users.

An additional available feature for the analysis is the possibility of grouping calculations together. This could for example connect the three calculations involved in a calculation of the chemisorption energy permanently. In the section "Grouping and Generation of Unique IDs" we shall describe how this works in more detail.

To illustrate the combined functionality of the CMR system, we show in Fig. 2 an overview of the system. The data may come from different electronic structure codes to be uploaded in the repository using the intermediate db-file format. During and after the upload, the data are analyzed by a set of agents based on the available taxonomy and folksonomy classification. The raw data and high-level data generated in tables can be accessed using the python, PHP, or the HTML/JavaScript interface.

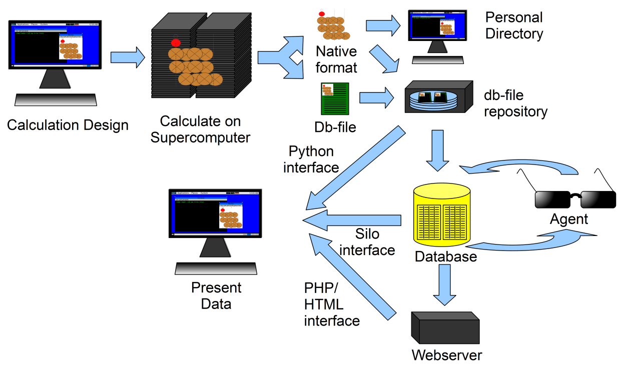


Figure 2: An overview of the CMR system. The main steps are the creation of the db-files, the uploading to the database, and the automatic analysis performed by the agents.

3 The inner workings

In the following we shall discuss some of the important details of the implementation of CMR. Most of the material here is not necessary to know about for the general or "casual" user of CMR, but is essential for the superuser who would like to tweak the system performance.

3.1 db-files and cmr-schemas

Db-files are stored in XML format. We chose XML[8] for several reasons: XML files can be easily extended and verified, the content can be read by humans, the structure is to a high degree self-explanatory, and there are xml-parsers available in almost any programming language. Anyone can therefore read db-files and might relatively easily create an application utilizing them independently of CMR.

Here is an extract of an XML file that contains the output from a GPAW[9] calculation:

```
...
 <calculator>
   <TotalEnergy><double>-586.442</double></TotalEnergy>
   <AtomicNumbers>
    <long_array length="2">
     <long>6</long>
     <long>8</long>
    </long_array>
   </AtomicNumbers>
 ...
 </calculator>
...
```

The content is intuitively obvious as follows: $TotalEnergy$ is equal -586.442 and $AtomicNumbers$ is an array containing 6 and 8. "double", "long_array" and "long" are the names of the variable types.

The difference between a db-file containing data from a GPAW calculation and one containing data from a different code, e.g. a Dacapo calculation, is defined by the schema. The schema declares the names of the variables, the types and whether they are optional or mandatory. The CMR schemas are used to validate the contents of the db-files. This enables early detection of possible problems with the data; if a field is expected to be an integer, but is a string, it would fail to be uploaded to the database.

3.2 Grouping and Generation of Unique IDs

The db-files are not only used to store converted information from programs. They are also used to store information about groups of calculations. The difference is only the schema. Using db-files, we are not only able to identify the members of a group but are also able to add keywords, a description, and user defined fields. An example would be the calculation of a chemisorption energy that needs three calculations. When creating this group, we add a field `chemisorption_energy` with the calculated result and also fields with the name of the surface atoms and the name of the adsorbate.

In a database, it is easy to know which items belong to a group, because every item has an automatically assigned unique id - but is it possible to transfer the data to a third party's database and keep the same id? No, because the third party has different data in the database, and hence the ids are most likely already used. To circumvent this issue, we create a hash value (unique identifier) from the content of the calculation. We use SHA1[10] with a key length of 160 bits, which is sufficiently long to be almost certain that no collisions will happen.

The hash and the ability to add user defined fields to groups open more options for exchanging data. The above described group containing the chemisorption energy can be transferred to the third party without including the db-files that were used to actually determine the chemisorption energy. This saves memory and since the group retains the reference (the unique ids of its members) it is at any time possible for the third party to identify the omitted parts and request them if necessary.

4 Software Design and Application of CMR

People using CMR have different needs: some want to use CMR to query their data, others would like to use the system to collect the data, but upload them to a database that arranges the data in a very specific way, while developers want to implement other readers for using their codes with CMR. To cope with these requirements CMR is built in a modular way and supports various groups of plug-ins that can easily be extended:

- converters: readers of foreign file formats that convert to db-files
- mappings: define mappings of names, types and units
- agents: autonomous or manually started processes that analyze data
- cmr-schemas: define exactly what data are to be collected from a reader and the types of the variables
- tests: tests a plug-in or any other functionality in CMR

Converting a supported output file (e.g. from Dacapo, Gaussian, GPAW and VASP) to a db-file is simple. Arguments, keywords and extra fields are placed in a python dictionary and passed as an argument to the convert function:

```
import cmr

params = {"input": "example.gpw",
          "db" True,
          "keywords": ["gpaw",
                       "example",
                       "test"],
          "files": ["log.txt"],
          "description": "Vacuum convergence test",
          "vacuum": 12.5
}

cmr.convert(params)
```

The flag `"db":True` means that the output should be written directly to the db-file repository. Every plug-in is expected to provide one or more **tests**. Tests are run when a release version is created or on demand by the user.

4.1 Screening example

During the CAMd Summer School 2008 at the Technical University of Denmark (DTU), a project was carried out with the aim of identifying ternary alkali-transition metal borohydrides for hydrogen storage.[11] During the school about 100 participants completed more than 5000 electronic structure calculations which were stored in an early version of CMR. The results are presented in Fig. 3 where the green box indicates the region of alloy stability and a decomposition energy which is advantageous for hydrogen storage.[11] All the data are available in our CMR at `https://cmr.fysik.dtu.dk/` together with data compiled in a project on solar-induced splitting of water.[12]

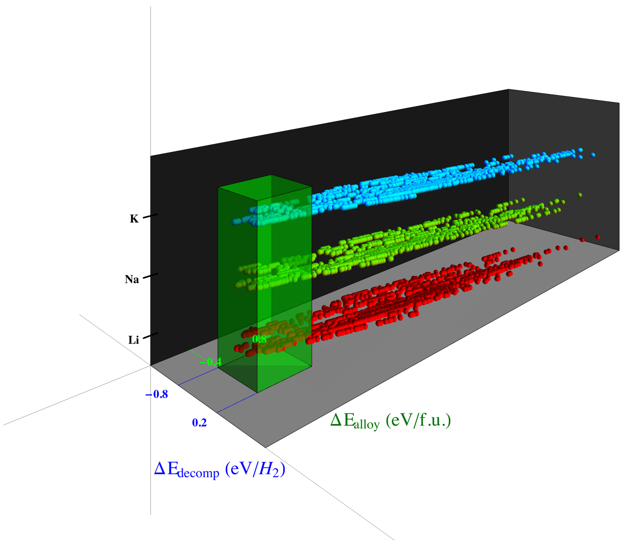


Figure 3: Energetic properties (alloy stability ($\Delta E_{\mathrm{alloy}}$) and decomposition energy ($\Delta E_{\mathrm{decomp}}$)) of (Li, Na, or K)-transition metal borohydrides calculated during the CAMd Summer School 2008.[11] The most promising candidates for reversible hydrogen storage are inside the box.

4.2 Competing Software and Availability

Recently a number of projects have shown up that go in a similar direction as CMR.[6, 14, 15] Quixote[16] for example is a system to organize, share and query data for computational chemistry codes. Like CMR it is distributed as open-source software, and implements a similar workflow to process data. The main advantages of CMR over Quixote are scriptable database access, possibility to add data to the database without a converter and being able to create groups of calculation with custom fields and keywords. Another example is the Materials Genome Project[13] that has the goal to provide a public database of electronic structure calculations for materials screening, structure prediction, analysis, and data mining. The unique feature of its closed source data generation framework is the combination of known existing compounds, automated DFT calculations, and analysis in order to predict novel materials.

CMR results from a collaboration under the Quantum Materials Informatics Project,[17] which aims at establishing the core technology for integrated computational materials design. Extensive information on CMR is available at `https://wiki.fysik.dtu.dk/cmr`.

5 Acknowledgments

We acknowledge financial support from the Danish Center for Scientific Computing and NABIIT. The Center for Atomic-scale Materials Design (CAMD) is funded by the Lundbeck Foundation. The SUNCAT Center for Interface Science and Catalysis is funded by the U.S. Department of Energy (DOE). J. G. acknowledges support from the Department of Energy, Office of Science, Office of Basic Energy Sciences, through the Early Career Research Program.

References

1. "MySQL." `http://www.mysql.com/`.
2. S. R. Bahn and K. W. Jacobsen, "An Object-Oriented Scripting Interface to a Legacy Electronic Structure Code," Computing in Science and Engineering, vol. 4, pp. 56–66, 2002.
3. "Atomic Simulation Environment (ASE)." `https://wiki.fysik.dtu.dk/ase/`.
4. "Python." `http://www.python.org/`.
5. "Electronic Structure Codes." `http://www.psi-k.org/codes.shtml`.
6. T. R. Munter, D. D. Landis, F. Abild-Pedersen, G. Jones, S. Wang, and T. Bligaard, "Virtual materials design using databases of calculated materials properties," Computational Science & Discovery, vol. 2, p. 015006, 2009.
7. "Wikipedia, the free encyclopedia." `http://en.wikipedia.org/wiki/Folksonomy`.
8. "Extensible Markup Language (XML)." `http://www.w3.org/standards/xml/`.
9. "Grid-based Projector-Augmented Wave method (GPAW)." `https://wiki.fysik.dtu.dk/gpaw/`.
10. "sha1." `http://en.wikipedia.org/wiki/SHA-1`.
11. J. S. Hummelshøj, "Density functional theory based screening of ternary alkali-transition metal borohydrides: A computational material design project," J. Chem. Phys., vol. 131, 2009.
12. I. E. Castelli, T. Olsen, S. Datta, D. D. Landis, S. Dahl, K. S. Thygesen, K. W. Jacobsen, "Computational screening of perovskite metal oxides for optimal solar light capture", Energy Environ. Sci., 2012, DOI: 10.1039/C1EE02717D
13. "A high-throughput infrastructure for density functional theory calculations," Comp. Mat. Sci vol. 50, pp. 2295-2310, 2011
14. "AflowLib." `http://www.aflowlib.org/`.
15. "ESTEST." `http://estest.ucdavis.edu/`.

16. S Adams P. de Castro P. Echenique, J. Estrada, M. Hanwell, P. Murray-Rust, P. Sherwood, J. Thomas, J. Townsend, The Quixote project: Collaborative and Open Quantum Chemistry data management in the Internet age, J. Cheminf., vol. 3, no. 1, p. 38, 2011.
17. "Quantum Materials Informatics Project." `http://www.qmip.org/`.

Appendix B

Computational Screening of Perovskite Metal Oxides for Optimal Solar Light Capture

I. E. Castelli, T. Olsen, S. Datta, D. D. Landis, K. S. Thygesen, S. Dahl, K. W. Jacobsen
Energy Environ. Sci., vol. 5, 2012

Energy & Environmental Science

Dynamic Article Links

Cite this: *Energy Environ. Sci.*, 2012, **5**, 5814

www.rsc.org/ees

PAPER

Computational screening of perovskite metal oxides for optimal solar light capture†

Ivano E. Castelli,[a] Thomas Olsen,[a] Soumendu Datta,[a] David D. Landis,[a] Søren Dahl,[b] Kristian S. Thygesen[a] and Karsten W. Jacobsen*[a]

Received 20th September 2011, Accepted 17th November 2011
DOI: 10.1039/c1ee02717d

One of the possible solutions to the world's rapidly increasing energy demand is the development of new photoelectrochemical cells with improved light absorption. This requires development of semiconductor materials which have appropriate bandgaps to absorb a large part of the solar spectrum at the same time as being stable in aqueous environments. Here we demonstrate an efficient, computational screening of relevant oxide and oxynitride materials based on electronic structure calculations resulting in the reduction of a vast space of 5400 different materials to only 15 promising candidates. The screening is based on an efficient and reliable way of calculating semiconductor band gaps. The outcome of the screening includes all already known successful materials of the types investigated plus some new ones which warrant further experimental investigation.

Introduction

The high living standard created in the world during the last century is to a large extent due to easy access to cheap fossil fuels. These resources are limited, and the ever increasing energy demands, together with the CO_2 related climate problems, make the development of sustainable energy technology one of the most important problems of today.[1] Direct harvesting and conversion of solar light to electrical energy in photovoltaic (PV) cells or to chemical energy by photoelectrochemical (PEC) reactions are the most obvious technologies to address this problem. Conventionally, both technologies rely on light collection in semiconductor (SC) materials with appropriate bandgaps that match the solar spectrum in order to obtain high energy conversion efficiency.[2–4] Here, we demonstrate an efficient, computational screening of relevant oxide and oxynitride materials based on electronic structure calculations showing that less than 1 out of 350 materials are realistic candidates for light-induced splitting of water.

The tremendous increase of computational power over the last couple of decades, in combination with methodological improvements, has made it possible to guide the development of new materials using first principles quantum mechanical calculations. Examples include the development of battery cathodes,[5] the construction of semiconductor superlattices,[6] searching for high stability alloys,[7] and, very recently, screening for high-performance piezoelectrics,[8] for organic photovoltaics[9,10] and for inorganic scintillator materials.[11] Here, we show that a newly

[a]*Center for Atomic-scale Materials Design, Department of Physics, Technical University of Denmark, DK - 2800 Kongens Lyngby, Denmark. E-mail: kwj@fysik.dtu.dk*
[b]*Center for Individual Nanoparticle Functionality, Department of Physics, Technical University of Denmark, DK - 2800 Kongens Lyngby, Denmark*

† Electronic supplementary information (ESI) available: Methods, Cubic Perovskite Oxides and Cubic Perovskite Oxinitrides sections; Table 1, 2, 3; Fig. 1, 2. See DOI: 10.1039/c1ee02717d

Broader context

For almost 40 years, researchers have tried to identify semiconductors suitable for photoelectrochemical water splitting under solar light. Investigations have focused on oxides and, more recently, on oxynitrides, due to their good properties with respect to stability. Inspired by this, we have performed a comprehensive computational screening of more than 5400 oxide/oxynitride compounds in the cubic perovskite structure covering 52 metals. The screening is based on criteria for stability and for the size and position of the bandgap. The material should allow for collecting a significant part of the solar photons and be able to drive the uphill water splitting reaction. The calculations of the bandgaps go beyond standard (semi-)local DFT and take into account explicitly the derivative discontinuity providing realistic estimates of the gaps. We end up with 10 oxides and 5 oxynitrides as candidates for light harvesting materials including $AgNbO_3$, $BaSnO_3$, $BaTaO_2N$, $CaTaO_2N$, $SrTaO_2N$, and $LaTiO_2N$, which are well known in the water splitting community. We suggest 9 new combinations for further experimental investigation.

implemented density functional method allows for completely new possibilities of screening material properties involving their band gaps.

The properties determining the usefulness of a SC material as light harvester in a PEC cell include[12,13] (i) a band gap allowing the utilization of a significant fraction of the solar spectrum; (ii) well positioned band edges relative to the water redox levels; (iii) high mobilities, allowing electrons and holes to reach the surface and reduce/oxidize the targets before recombining, and (iv) chemical/structural stability under irradiation. In addition, low cost and non-toxicity are necessary properties. Numerous efforts have been made to find an efficient material for splitting water into H_2 and O_2 under visible light irradiation going more than 40 years back to Honda and Fujishima's report on electrochemical photolysis using TiO_2,[14] but so far the ideal material has not been found.[3]

Here, we focus mainly on aspects (i), (ii) and (iv) mentioned above, namely the search for stable materials with optimal, well positioned, bandgaps. We consider metal oxides and oxynitrides, due to their high stability, and we concentrate on the cubic perovskite structure with general formula ABO_3 (space group $Pm\bar{3}m$), due to the large variety of properties and applications of materials in this structure.[15] We first consider the binary oxides where much experimental information is already available. The screening method is then applied to binary oxynitrides which generally have better positioning of the bandgap for water splitting compared to the oxides, but where much less experimental information is available, making theoretical screening necessary. Our study points to six new oxides and one oxynitride candidate for water splitting which should warrant experimental investigation.

Results and discussion

The first step is to find and validate an appropriate method for calculation of oxide and oxynitride stabilities and bandgaps. With respect to the stability, we use a standard DFT-GGA in the form of the RPBE-functional.[16] (Details of the methods used in this paper can be found in the Methods section in the ESI †).

Reliable calculations of the bandgaps require a density functional beyond GGA. We use the so-called GLLB-SC functional[17,18] which is demonstrated in Fig. 1 to predict the magnitudes of the bandgaps of a selection of non-magnetic metal oxides with different equilibrium structures[19] within an absolute deviation of 0.5 eV—an accuracy sufficient for the present screening study. The computational cost of DFT-GLLB-SC is significantly lower than for many-body perturbation techniques such as the GW approximation and is crucial for the success of the screening. We use the GPAW code[20,21] for all calculations presented in the following.

A cubic perovskite (see structure in Fig. 2C) consists of large 12-coordinated cations at the so-called A sites and small 6-coordinated cations at the B sites. Compounds with different combinations of cation charges in the A and B sites, *e.g.* 1 + 5, 2 + 4, and 3 + 3, have been found in nature. We consider all the possible combinations of perovskites obtained starting from the non-radioactive metals of the periodic table.

We define the formation energy, ΔE, of the perovskite metal oxides as the energy difference in the following reaction:

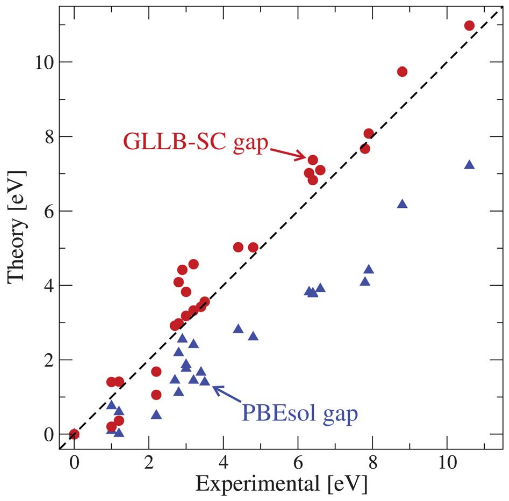


Fig. 1 DFT calculated bandgaps of selected oxides. Comparison between the theoretical and experimental bandgap of non-magnetic metal oxides in their most stable structure. The gaps are calculated using both the standard PBEsol (blue triangles) and the GLLB-SC functional (red circles). The dashed line represents the perfect matching between experiments and theory. (Details of the calculations with a list of the calculated oxides can be found in Table 1 of the ESI†).

$$A(s) + B(s) + 3H_2O(g) \rightarrow ABO_3(s) + 3H_2(g). \qquad (1)$$

We use water and H_2 as reference for O_2 instead of molecular oxygen, because the material we are looking for has to work in an aqueous environment. This choice is conservative with respect to O_2 because water is more stable than molecular oxygen and hydrogen by 2.46 eV per water molecule. The reaction energy is calculated directly from the DFT total energies of the participating molecules and solids. We estimate the Gibbs free energy of the reaction with water in the liquid phase following Nørskov *et al.*[22] to be within 0.1 eV of the calculated DFT total energy difference. We therefore simply use ΔE for the perovskite oxide to estimate the stability relative to the two metals in their most stable structures.

Fig. 2A summarizes the results for the formation energies per atom and bandgaps for the 2704 investigated oxides in the perovskite structure. In the figure, the square corresponding to a given oxide containing two metals is split into two parts with the lower, left triangle indicating the stability (from red to blue with decreasing stability) and the upper, right triangle the bandgap (Fig. 2B). The data are available in the database *Computational Materials Repository*,[23] developed at CAMD, at the web address http://cmr.fysik.dtu.dk/.

The stability of a compound can be seen to be the result of three factors: (i) the sum of the possible oxidation numbers of the two metals has to be equal to 6 since the three oxygen atoms in the unit cell require 2 electrons each in order to form a compound without free charge; (ii) the radii of the A and B ions have to be in reasonable proportions and (iii) elements with low electronegativity are preferable for forming bonds with oxygen. The last

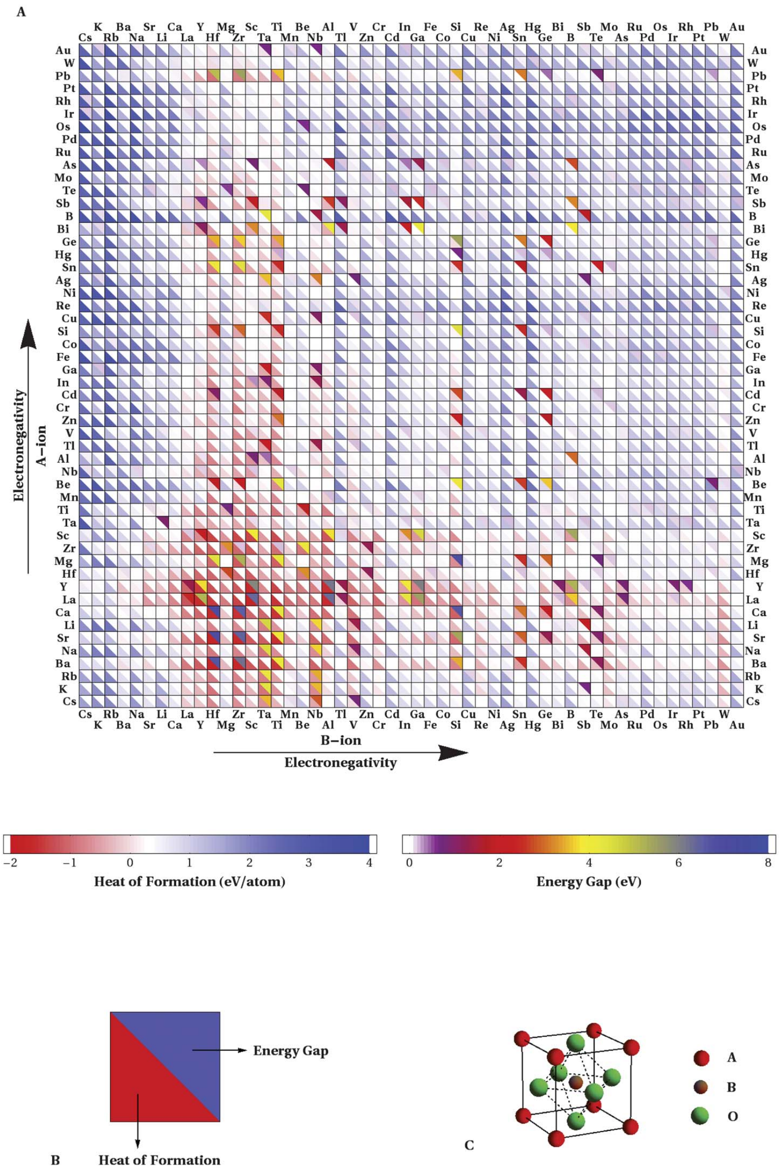


Fig. 2 (A) DFT calculated heat of formations per atom and bandgaps of perovskite binary metal oxides. (B) Each square represents an oxide with the lower, left triangle showing the formation energy with red indicating stability (and blue instability) while the upper, right triangle showing the bandgap with red indicating an advantageous bandgap in the range 1.5–3.0 eV. White indicates zero bandgap, purple indicates too small a gap, while yellow or blue indicates bandgaps larger than 3 eV. The pure chemical elements are sorted for increasing electronegativity. (C) reports the unit cell of the cubic perovskite structure. Data available at the web address: http://cmr.fysik.dtu.dk/.

factor is more relevant for the atoms in the A site due to the nonequivalence of the A and B ion positions.

The second design criterion we focus on is the size of the bandgap which we require to be in the range 1.5 eV to 3 eV. The lower limit comes about as the water-splitting threshold of 1.23 eV plus ∼0.25 eV to account for the electrochemical overpotentials.[3] A more realistic limit may be even higher since the splitting of the quasi-Fermi level is smaller than the gap when the SC is under illumination.[24] However, for tandem cells the lower bandgap limit is relevant.[3] Beyond the higher limit of 3.0 eV too little of the solar spectrum is left to be of interest. Depending on construction of the solar cell device, the light capture in a cell may be thin or thick and we therefore perform the search for either the direct or the indirect gap. The color scale applied for the bandgap in Fig. 2 is chosen so that red indicates a gap in the design window. Compounds with good stabilities and gaps can thus be spotted as red squares.

We note that in accordance with Aguiar *et al.*,[25] the bandgap is seen to decrease when increasing the electronegativity of the B ion or when increasing the crystallographic symmetry by adjusting the size of the A ion.

The stabilities and the bandgaps are somewhat correlated as can be seen in Fig. 3, where the pale orange area indicates the region we are interested in. It is a challenge to combine a small gap with a high stability. However, quite a few of the compounds with very small or zero bandgap also exhibit high stability. To this group belong all the perovskites with an odd number of electrons for which the bands at the Fermi level are not completely filled or empty even if considering the possibility of a spin up and spin down occupation.

At this stage the screening identifies 43 binary oxides which fulfill the two design criteria: $\Delta E < 0.2$ eV/atom and $1.5 <$ bandgap < 3.0 eV where we allow for a small positive energy of 0.2 eV/atom to allow for mildly metastable compounds. Many of the resulting candidates are in fact unstable towards a combination of restructuring and decomposition and we therefore expand our pool of reference systems used to assess stability to include not only the bulk metals but also the most stable single- and bi-metal oxides in their equilibrium structures as listed in the ICSD[19] and the Materials Genome[26,27] databases.

An additional criteria to stability and bandgap is the position of the band edges: for evolving both hydrogen and oxygen, the calculated gap should straddle both the hydrogen and oxygen evolution potentials (horizontal lines in Fig. 4). To estimate the band edges, we use an empirical method suggested and investigated by Butler and Ginley[28] and validated by Xu and Schoonen.[29] The scheme proceeds by positioning the middle of the gap at $E_0 + (\chi_A\chi_B\chi_O^3)^{1/5}$, where E_0 is the difference between the normal hydrogen electrode level and vacuum ($E_0 = -4.5$ eV) and χ_M denotes the electronegativity of the neutral atom M in the Mulliken scale, and the two edges are obtained by adding or subtracting half of the gap.

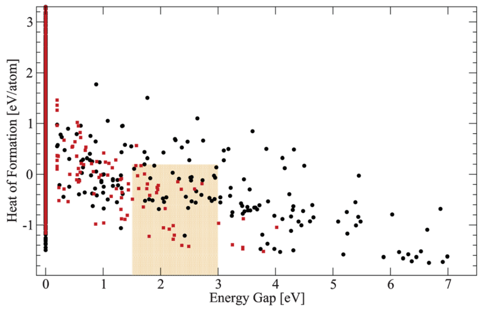


Fig. 3 Correlation between the heat of formation per atom and the bandgap for the oxide (black circles) and oxynitride (red squares) compounds. The region for candidates for solar light harvesting corresponds to the orange area.

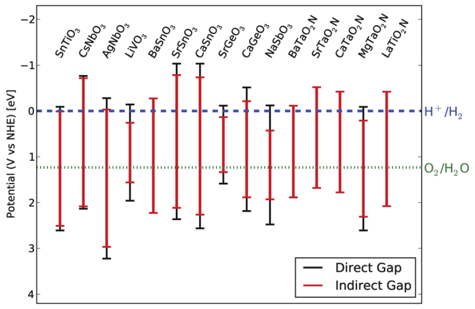


Fig. 4 The identified oxides and oxynitrides in the cubic perovskite structure with potential for splitting water in visible light. The figure shows the calculated band edges for both the direct (red) and indirect (black) gaps. The levels for hydrogen and oxygen evolution are also indicated.

After considering the more stringent stability criterion and the positions of the band edges the procedure results in only 10 binary oxides as candidate materials as indicated in the left part of Fig. 4. (A more detailed list is found in Table 2 and Fig. 1 of the ESI†). Some of these compounds are actually known to exist in other periodic structures, but including those in the pool of references does not change the list of candidates. However, two of the materials ($SrSnO_3$ and $CaSnO_3$) undergo lattice distortions and thereby obtain larger gaps beyond the visible light absorption limit. Two compounds are already known in the cubic perovskite structure: $AgNbO_3$ and $BaSnO_3$. Of these, $AgNbO_3$ is well-known to split water in visible light in the presence of sacrificial reagents,[30] while $BaSnO_3$ performs less well because of defect-assisted recombination.[31] It can be noted that if we relax the criterion on the bandgap we find oxides which can split water in UV light. 10 materials, like $AgTaO_3$ and $SrTiO_3$, which are well known to split water in UV light,[12] have a gap in a range between 3 and 4 eV. To our knowledge no other cubic perovskites that can split water in visible light have been identified, and we therefore conclude that the screening procedure performs well and we turn our attention to the more unexplored territory of oxynitrides.

The oxynitrides are especially interesting from the point of view of the gap position relative to the energy levels for hydrogen and oxygen evolution. This is because the valence band (VB) edge is usually dominated by N p-orbitals which are higher in energy than the O p-orbitals, that mainly compose the VB of the oxides.

Using the same approach as for the oxides, we screen the possible combinations of two metals in the oxynitride cubic perovskite structure (ABO_2N) using the same three design criteria as for the oxides, where we now also include the most stable single- and bi-metal nitrides (M_xN_y and $M^1_xM^2_yN_z$) and the single-metal oxynitrides ($M_xN_yO_z$) in the pool of reference systems. The chemical potential of a nitrogen atom is taken from the nitrogen molecule.

Fig. 5 reports the results for the formation energies per atom and bandgaps for the 2704 oxynitrides with the cubic perovskite structure. As also shown in Fig. 3, the oxygen substitution is followed by a general reduction in the size of the bandgap. The

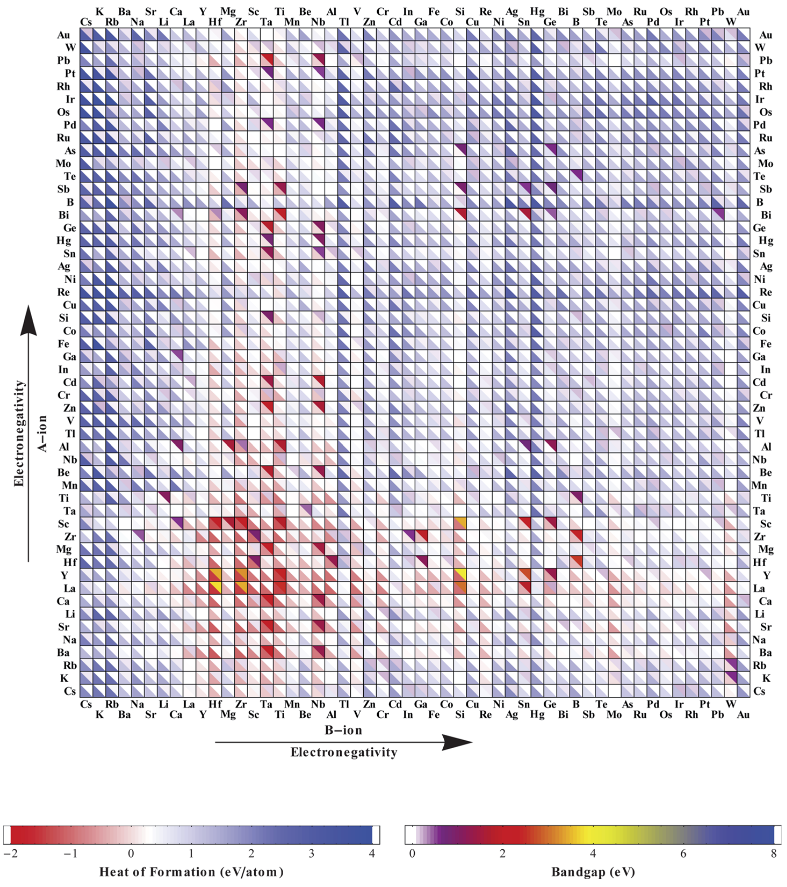


Fig. 5 DFT calculated heat of formations per atom and bandgaps of perovskite binary metal oxynitrides. The color bars are the same as used in Fig. 2.

effect of more general anion substitution especially in relation to the size and the position of the bandgap is relevant for the design of new materials able to split water. An investigation of those effects will be performed in the future.

The resulting five best candidates are shown in Fig. 4 and in more detail in Table 3 and Fig. 2 of the ESI.† Four of these combinations are already known ($BaTaO_2N$, $SrTaO_2N$, $CaTaO_2N$ and $LaTiO_2N$) and perform well for hydrogen evolution.[32] In fact, these compounds are, to our knowledge, the only cubic perovskite oxynitrides which have been shown experimentally to split water. We take this as a strong validation of our approach. The last compound, $MgTaO_2N$, has not yet been investigated experimentally.

Conclusions

In summary we have demonstrated that fast computational screening with respect to stability and bandgap is an efficient way to discover new light harvesting materials for water splitting. The method is based on a special exchange-correlation functional that produces sufficiently reliable bandgaps at low computational cost. The method is verified by screening 2704 oxides with the cubic perovskite structure in order to find the best candidate for photoelectrolytic hydrogen production by water splitting. Ten materials, of which two are already known, fulfilled the requirements set up in the screening. This remarkable result shows the strength of the screening approach. We have continued with screening of oxynitrides in the same structure and found five possible candidates of which four are already known. To the best of our knowledge, the set of 15 candidates coming out from our screening includes all the compounds in the cubic perovskite structure that are known to be suitable for water splitting. It seems natural to move forward with the method for other materials that are relevant for photocatalytic water splitting or other related technologies, like thin film solar cells.

Acknowledgements

The authors acknowledge support from the Danish Center for Scientific Computing through grant HDW-1103-06, from the Catalysis for Sustainable Energy initiative funded by the Danish Ministry of Science, Technology and Innovation, and from the

Danish Council for Strategic Research's Programme Commission on Strategic Growth-Technologies (NABIIT). The Center for Atomic-scale Materials Design is sponsored by the Lundbeck Foundation.

References

1 N. Lewis and D. Nocera, *Proc. Natl. Acad. Sci. U. S. A.*, 2006, **103**, 15729–15735.
2 W. Shockley and H. J. Queisser, *J. Appl. Phys.*, 1961, **32**, 510–519.
3 M. G. Walter, E. L. Warren, J. R. McKone, S. W. Boettcher, Q. Mi, E. A. Santori and N. S. Lewis, *Chem. Rev.*, 2010, **110**, 6446–6473.
4 M. Gratzel, *Nature*, 2001, **414**, 338–344.
5 G. Ceder, Y.-M. Chiang, D. R. Sadoway, M. K. Aydinol, Y.-I. Jang and B. Huang, *Nature*, 1998, **392**, 694.
6 A. Franceschetti and A. Zunger, *Nature*, 1999, **402**, 60.
7 G. H. Johannesson, T. Bligaard, A. V. Ruban, H. L. Skriver, K. W. Jacobsen and J. K. Nørskov, *Phys. Rev. Lett.*, 2002, **88**, 255506.
8 R. Armiento, B. Kozinsky, M. Fornari and G. Ceder, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2011, **84**, 014103.
9 J. Hachmann, R. Olivares-Amaya, S. Atahan-Evrenk, C. Amador-Bedolla, R. S. Sanchez-Carrera, A. Gold-Parker, L. Vogt, A. M. Brockway and A. Aspuru-Guzik, *J. Phys. Chem. Lett.*, 2011, **2**, 2241–2251.
10 N. M. O'Boyle, C. M. Campbell and G. R. Hutchison, *J. Phys. Chem. C*, 2011, **115**, 16200–16210.
11 W. Setyawan, R. M. Gaume, S. Lam, R. S. Feigelson and S. Curtarolo, *ACS Comb. Sci.*, 2011, **13**, 382–390.
12 A. Kudo and Y. Miseki, *Chem. Soc. Rev.*, 2009, **38**, 253–278.
13 X. Chen, S. Shen, L. Guo and S. S. Mao, *Chem. Rev.*, 2010, **110**, 6503–6570.
14 A. Fujishima and K. Honda, *Nature*, 1972, **238**, 37–38.
15 T. Ishihara, *Perovskite Oxide for Solid Oxide Fuel Cells*, Springer Verlag, 2009.
16 B. Hammer, L. B. Hansen and J. K. Nørskov, *Phys. Rev. B: Condens. Matter*, 1999, **59**, 7413–7421.
17 O. Gritsenko, R. van Leeuwen, E. van Lenthe and E. J. Baerends, *Phys. Rev. A: At., Mol., Opt. Phys.*, 1995, **51**, 1944.
18 M. Kuisma, J. Ojanen, J. Enkovaara and T. T. Rantala, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2010, **82**, 115106.
19 *ICSDWeb*, http://www.fiz-karlsruhe.de/icsd_web.html.
20 J. J. Mortensen, L. B. Hansen and K. W. Jacobsen, *Phys. Rev. B: Condens. Matter Mater. Phys.*, 2005, **71**, 35109.
21 J. Enkovaara, C. Rostgaard, J. J. Mortensen, J. Chen, M. Dulak, L. Ferrighi, J. Gavnholt, C. Glinsvad, V. Haikola, H. A. Hansen, H. H. Kristoffersen, M. Kuisma, A. H. Larsen, L. Lehtovaara, M. Ljungberg, O. Lopez-Acevedo, P. G. Moses, J. Ojanen, T. Olsen, V. Petzold, N. A. Romero, J. Stausholm-Møller, M. Strange, G. A. Tritsaris, M. Vanin, M. Walter, B. Hammer, H. Hakkinen, G. K. H. Madsen, R. M. Nieminen, J. K. Nørskov, M. Puska, T. T. Rantala, J. Schiotz, K. S. Thygesen and K. W. Jacobsen, *J. Phys.: Condens. Matter*, 2010, **22**, 253202.
22 J. Nørskov, J. Rossmeisl, A. Logadottir, L. Lindqvist, J. Kitchin, T. Bligaard and H. Jonsson, *J. Phys. Chem. B*, 2004, **108**, 17886–17892.
23 *Computational Materials Repository*, https://wiki.fysik.dtu.dk/cmr/ (software), and https://cmr.fysik.dtu.dk/ (database).
24 M. F. Weber and M. J. Dignam, *Int. J. Hydrogen Energy*, 1986, **11**, 225–232.
25 R. Aguiar, D. Logvinovich, A. Weidenkaff, A. Rachel, A. Reller and S. G. Ebbinghaus, *Dyes Pigm.*, 2008, **76**, 70–75.
26 *Materials Genome*, http://www.materialsgenome.org/.
27 A. Jain, G. Hautier, C. J. Moore, S. P. Ong, C. C. Fischer, T. Mueller, K. A. Persson and G. Ceder, *Comput. Mater. Sci.*, 2011, **50**, 2295–2310.
28 M. A. Butler and D. S. Ginley, *J. Electrochem. Soc.*, 1978, **125**, 228–232.
29 Y. Xu and M. A. Schoonen, *American Mineralogist*, 2000, **85**, 543–556.
30 H. Kato, H. Kobayashi and A. Kudo, *J. Phys. Chem. B*, 2002, **106**, 12441–12447.
31 W. Zhang, J. Tang and J. Ye, *J. Mater. Res.*, 2007, **22**, 1859–1871.
32 D. Yamasita, T. Takata, M. Hara, J. Kondo and K. Domen, *Solid State Ionics*, 2004, **172**, 591–595.

Appendix C

Density functional theory based screening of ternary alkali-transition metal borohydrides: A computational material design project

Hummelshøj, J. S. and Landis, D. D. and Voss, J. and Jiang, T. and Tekin, A. and Bork, N. and Dułak, M. and Mortensen, J. J. and Adamska, L. and Andersin, J. and Baran, J. D. and Barmparis, G. D. and Bell, F. and Bezanilla, A. L. and Bjork, J. and Björketun, M. E. and Bleken, F. and Buchter, F. and Bürkle, M. and Burton, P. D. and Buus, B. B. and Calborean, A. and Calle-Vallejo, F. and Casolo, S. and Chandler, B. D. and Chi, D. H. and Czekaj, I. and Datta, S. and Datye, A. and Delariva, A. and Despoja, V. and Dobrin, S. and Engelund, M. and Ferrighi, L. and Frondelius, P. and Fu, Q. and Fuentes, A. and Fürst, J. and García-Fuente, A. and Gavnholt, J. and Goeke, R. and Gudmundsdottir, S. and Hammond, K. D. and Hansen, H. A. and Hibbitts, D. and Hobi, E. and

Howalt, J. G. and Hruby, S. L. and Huth, A. and Isaeva, L. and Jelic, J. and Jensen, I. J. T. and Kacprzak, K. A. and Kelkkanen, A. and Kelsey, D. and Kesanakurthi, D. S. and Kleis, J. and Klüpfel, P. J. and Konstantinov, I. and Korytar, R. and Koskinen, P. and Krishna, C. and Kunkes, E. and Larsen, A. H. and Lastra, J. M. G. and Lin, H. and Lopez-Acevedo, O. and Mantega, M. and Martínez, J. I. and Mesa, I. N. and Mowbray, D. J. and Mýrdal, J. S. G. and Natanzon, Y. and Nistor, A. and Olsen, T. and Park, H. and Pedroza, L. S. and Petzold, V. and Plaisance, C. and Rasmussen, J. A. and Ren, H. and Rizzi, M. and Ronco, A. S. and Rostgaard, C. and Saadi, S. and Salguero, L. A. and Santos, E. J. G. and Schoenhalz, A. L. and Shen, J. and Smedemand, M. and Stausholm-Møller, O. J. and Stibius, M. and Strange, M. and Su, H. B. and Temel, B. and Toftelund, A. and Tripkovic, V. and Vanin, M. and Viswanathan, V. and Vojvodic, A. and Wang, S. and Wellendorff, J. and Thygesen, K. S. and Rossmeisl, J. and Bligaard, T. and Jacobsen, K. W. and Nørskov, J. K. and Vegge, T.

J. Chem. Phys., vol. 131, Jul7 2009

Density functional theory based screening of ternary alkali-transition metal borohydrides: A computational material design project

J. S. Hummelshøj, D. D. Landis, J. Voss, T. Jiang, A. Tekin, N. Bork, M. Dułak, J. J. Mortensen, L. Adamska, J. Andersin, J. D. Baran, G. D. Barmparis, F. Bell, A. L. Bezanilla, J. Bjork, M. E. Björketun, F. Bleken, F. Buchter, M. Bürkle, P. D. Burton, B. B. Buus, A. Calborean, F. Calle-Vallejo, S. Casolo, B. D. Chandler, D. H. Chi, I Czekaj, S. Datta, A. Datye, A. DeLaRiva, V Despoja, S. Dobrin, M. Engelund, L. Ferrighi, P. Frondelius, Q. Fu, A. Fuentes, J. Fürst, A. García-Fuente, J. Gavnholt, R. Goeke, S. Gudmundsdottir, K. D. Hammond, H. A. Hansen, D. Hibbitts, E. Hobi, Jr., J. G. Howalt, S. L. Hruby, A. Huth, L. Isaeva, J. Jelic, I. J. T. Jensen, K. A. Kacprzak, A. Kelkkanen, D. Kelsey, D. S. Kesanakurthi, J. Kleis, P. J. Klüpfel, I Konstantinov, R. Korytar, P. Koskinen, C. Krishna, E. Kunkes, A. H. Larsen, J. M. G. Lastra, H. Lin, O. Lopez-Acevedo, M. Mantega, J. I. Martínez, I. N. Mesa, D. J. Mowbray, J. S. G. Mýrdal, Y. Natanzon, A. Nistor, T. Olsen, H. Park, L. S. Pedroza, V Petzold, C. Plaisance, J. A. Rasmussen, H. Ren, M. Rizzi, A. S. Ronco, C. Rostgaard, S. Saadi, L. A. Salguero, E. J. G. Santos, A. L. Schoenhalz, J. Shen, M. Smedemand, O. J. Stausholm-Møller, M. Stibius, M. Strange, H. B. Su, B. Temel, A. Toftelund, V Tripkovic, M. Vanin, V Viswanathan, A. Vojvodic, S. Wang, J. Wellendorff, K. S. Thygesen, J. Rossmeisl, T. Bligaard, K. W. Jacobsen, J. K. Nørskov, and T. Vegge[a)]
The 2008 CAMD Summer School in Electronic Structure Theory and Materials Design, Center for Atomic-scale Materials Design, Department of Physics, Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark[b)]

(Received 20 November 2008; accepted 8 May 2009; published online 1 July 2009)

We present a computational screening study of ternary metal borohydrides for reversible hydrogen storage based on density functional theory. We investigate the stability and decomposition of alloys containing 1 alkali metal atom, Li, Na, or K (M_1); and 1 alkali, alkaline earth or $3d/4d$ transition metal atom (M_2) plus two to five $(BH_4)^-$ groups, i.e., $M_1M_2(BH_4)_{2-5}$, using a number of model structures with trigonal, tetrahedral, octahedral, and free coordination of the metal borohydride complexes. Of the over 700 investigated structures, about 20 were predicted to form potentially stable alloys with promising decomposition energies. The $M_1(Al/Mn/Fe)(BH_4)_4$, $(Li/Na)Zn(BH_4)_3$, and $(Na/K)(Ni/Co)(BH_4)_3$ alloys are found to be the most promising, followed by selected $M_1(Nb/Rh)(BH_4)_4$ alloys. © *2009 American Institute of Physics*.
[DOI: 10.1063/1.3148892]

I. INTRODUCTION

The development of sustainable energy solutions for the future requires new and improved materials. Specifically designed material properties are needed to solve the grand challenges in energy production, storage, and conversion. Within energy storage, hydrogen has been investigated extensively over the past decade[1] as one of the few promising energy carriers which can provide a high energy density without resulting in CO_2 emission by the end user. Finding materials for efficient, reversible hydrogen storage, however, remains challenging. Here, the specific requirements of the rapidly growing transportation sector coupled with complex engineering challenges[2] have directed research toward complex materials with extreme hydrogen storage capacities[3] such as metal borohydrides[4] and metal ammines.[5] Finding materials with high reversible hydrogen content and optimal thermodynamic stability is essential if hydrogen is going to be used as a commercial fuel in the transport sector. The binary metal borohydrides have been studied extensively: the alkali based compounds, e.g., $LiBH_4$,[6–8] are too thermodynamically stable, the alkaline earth compounds are kinetically too slow and practically irreversible,[9] and the transition metal borohydrides are either unstable or irreversible.[10] This leaves hope that mixed metal ("alloyed") systems might provide new opportunities.

The use of computational screening techniques has proved a valuable tool in narrowing the phase space of potential candidate materials for hydrogen storage.[11,12] Recent density functional theory (DFT) calculations have shown that the thermodynamic properties of even highly complex borohydride superstructures can be estimated by DFT using simple model structures, if the primary coordination polyhedra are correctly accounted for.[13] These findings enable faster screening studies of thermodynamic stability and decomposition temperatures for, e.g., ternary and quaternary borohydride systems; not only in terms of reduced computational effort due to smaller system sizes but also with the advantage that the exact space group does not need to be known

[a)]Electronic mail: teve@risoe.dtu.dk.
[b)]For a full list of affiliations, see Ref. 41.

a priori.

In the present paper, we apply a "local coordination screening" (LCS) approach to search for novel metal borohydrides. The vast majority of the calculations were performed as part of the 2008 CAMD summer school in electronic structure theory and materials design, where more than 100 scientists combined DFT calculations, database methods, and screening techniques to investigate the structure and stability of promising ternary borohydrides. A few additional calculations were subsequently performed based on the insight gained from the initial screening.

Out of 757 investigated $M_1M_2(BH_4)_{2-5}$ (M_1=alkali metal and M_2=alkali, alkaline earth or $3d/4d$ transition metal) compositions and structures, a total of 22 were found to form potentially stable alloys with promising decomposition energies, which should subsequently be subjected to more detailed theoretical and experimental verification.

II. COMPUTATIONAL SETUP

Groups of alloy compositions and structures were divided among different groups of scientists, each of which was responsible for its own subset of the alloy configuration space. A number of predefined structural templates and optimization procedures had been prepared to assist the groups in setting up structures and calculations for the initial optimization (see Sec. II B). This was done to ensure a sufficient accuracy in all calculations (i.e., convergence with respect to plane wave cutoff, k-point sampling, etc.).

To ensure reliability of the generated results, an automated checking procedure was enforced before a result could be included in the database (see Sec. III) to ensure the presence of the required output (total energies, lattice constants, etc.).

A. Computational parameters

The total energies and gradients were calculated within density functional theory[14] as implemented by the software package Dacapo.[15] A plane wave basis set with a cutoff energy of 350 eV (density grid cutoff of 700 eV) and the RPBE exchange-correlation functional[15] were used for all calculations. Dacapo uses ultrasoft pseudopotentials[16] for a description of the ionic cores. The coordinate optimization was implemented and performed within the atomic simulation environment.[17] The electronic Brillouin zones were sampled with $(4\times4\times4)$ k-points (spacings of ~0.05 Å^{-1}). A quasi-Newton method[18] was used for all relaxations.

B. Configuration space and template structures

The alloys which were initially screened have the general formula $M_1M_2(BH_4)_x$, where $M_1\in\{\mathrm{Li,Na,K}\}$ and $x=2-4$. The $x=2$ alloys were investigated for $M_2\in\{\mathrm{Li,Na,K}\}$, and $x=3,4$ for $M_2\in\{\mathrm{Li,Na,K,Mg,Al,Ca,Sc-Zn,Y-Mo,Ru-Cd}\}$.

In order to limit the total number of calculations, only template structures with tetrahedral and octahedral coordination of the $(BH_4)^-$ groups to the metal atoms were used. Most metals prefer an octahedral coordination of their ligands, but for the metal borohydrides the ligand-ligand repulsion between the relatively large $(BH_4)^-$ ions often forces a lower coordination number. The primary structures observed and reported in literature for the alkali and alkaline earth borohydrides are either tetrahedral (for the smallest Li and Mg) or octahedral (for the larger Na, K, and Ca), while a trigonal planar ligand arrangement is observed for $Al(BH_4)_3$. However, Al can also have a tetrahedral coordination as is the case of the $LiAl(BH_4)_4$ alloy obtained here (see Sec. V), and since the radii of the considered ions lie between the radius for K and the radius for Al, the tetrahedral and octahedral primary structures are expected to be representative.

For each alloy composition, four different template structures were used to sample the tetrahedral and octahedral primary structures in the combinations: tetrahedral/tetrahedral, octahedral/octahedral, tetrahedral/octahedral, and octahedral/tetrahedral, referring to the coordination of the $(BH_4)^-$ groups to the M_1 and M_2 atoms, respectively. The coordination polyhedra were either corner sharing, edge sharing, or a combination to yield the required stoichiometric ratio of $(BH_4)^-$ groups (see Fig. 1). All structures were designed to have a unit cell containing only one formula unit (see Sec. II D). It has previously been shown that these simple template structures can be within ~0.1 eV (10 kJ/mol H_2) of the true ground state energy if the local coordination is correctly accounted for; e.g., $M_1M_2(BH_4)_2$-tetra for $LiBH_4$,[7] $M_1M_2(BH_4)_4$-octa for $Ca(BH_4)_2$,[19] and even $M_1M_2(BH_4)_4$-tetra for the free energy of $Mg(BH_4)_2$ superstructures.[13]

The initial optimization of the structures only relaxed the hydrogen positions and the unit cell volume while keeping the metal-boron coordination polyhedra fixed. For a given set of (M_1,M_2), the most stable structure was then used as the starting point for a calculation in which all atomic positions and the unit cell were relaxed. Even though many of the structures did not change significantly during the final relaxation, it added, in principle, an additional structure to the phase space for each set of (M_1,M_2). These are included as "other" structures in the results (Figs. 3–12) to distinguish them from the structures with fixed metal-boron coordination polyhedra, even though the original coordination polyhedra are only slightly distorted in many of them.

A number of structures were subsequently added based on the knowledge gained from the initial screening and the reference binary borohydride structures (see Secs. IV and VI). In some of these structures, the metal ions had the same valence as in the reference structures, which meant that the four $x=2$ templates were also applied to $M_2\in\{\mathrm{Ni,Pd,Cu,Ag}\}$, while a new template for $x=5$ was investigated for $M_2\in\{\mathrm{Ti,Zr}\}$ in the two combinations tetrahedral/octahedral and octahedral/tetrahedral. An alternative $x=3$ tetragonal/trigonal template was applied to $M_2\in\{\mathrm{Mg,Al,Ca,Sc-Zn,Y-Mo,Ru-Cd}\}$ to investigate possible size effects. In this structure, the M_1 ion has a tetrahedral coordination while the M_2 atom is surrounded by three $(BH_4)^-$ groups in a trigonal planar arrangement (see Fig. 2). This enabled the metal-boron distances for the two metals to be optimized independently, which was not pos-

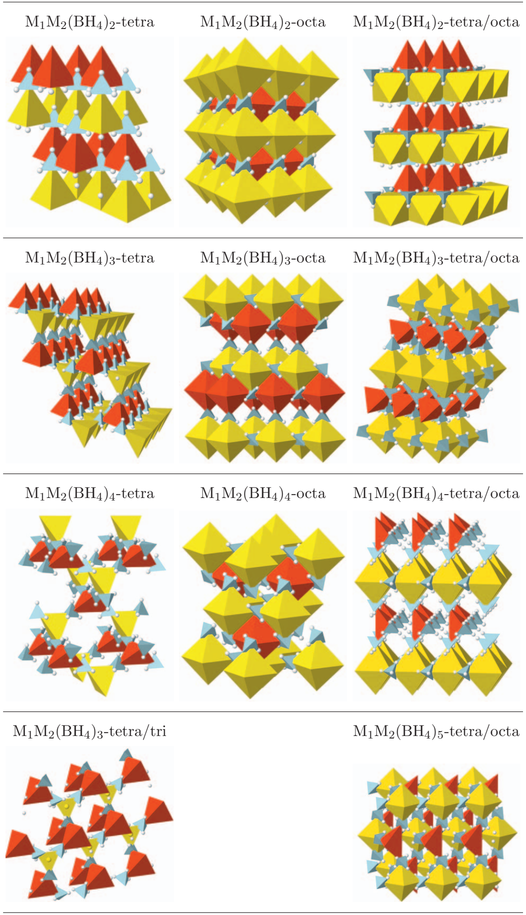


FIG. 1. The template structures of $M_1M_2(BH_4)_{2-5}$. Red and yellow polyhedra show the coordination of the B atoms around the M_1 and M_2 atoms, respectively; blue tetrahedra represent the $(BH_4)^-$ groups. The octa/tetra structures are obtained by switching M_1 and M_2 in the tetra/octa structures.

sible in the original $x=3$ templates, but found to be required to obtain the preferred local coordination of certain alloys.

In total, 757 structures have been simulated and are reported herein.

C. Group calculations

The 69 sets of (M_1, M_2) combinations investigated in this study were divided among 32 groups of scientists for the initial screening. Each group followed step I of the calculational procedure outlined below for each alloy containing M_1 and M_2 and step II for the most stable resulting structure.

D. Calculational procedure

1. Step I

An initial structure was set up by calling a function that populates one of the four template structures with two supplied metal ions, e.g., Li and Sc. The function utilizes the ionic radii obtained from the calculations of binary reference borohydrides, i.e., individual metal atom borohydrides, to calculate metal-boron distances, where the ionic radius used for a $(BH_4)^-$ group depends on whether a face, edge or corner of the H-tetrahedron points toward the metal atom. In general, this ensured that the effective lattice constant and the c/a ratio were close to the optimum. The initial structure was used as the initial guess for the first iteration of the following procedure.

TABLE I. The calculated reference energies for the binary borohydrides in their most stable template structures (see Fig. 2).

	wt % (kg H_2/kg material)	ΔE_{decomp} (eV/H_2)
$K(BH_4)$	7.5	−0.968
$Na(BH_4)$	10.7	−0.729
$Li(BH_4)$	18.5	−0.422
$Ag(BH_4)$	3.3	0.278
$Cu(BH_4)$	5.1	0.352
$Pd(BH_4)$	3.3	0.661
$Ni(BH_4)$	5.5	0.680
$Ca(BH_4)_2$	11.6	−0.636
$Mg(BH_4)_2$	14.9	−0.467
$Zn(BH_4)_2$	8.5	−0.063
$Cd(BH_4)_2$	5.7	−0.043
$V(BH_4)_2$	10.0	−0.031
$Nb(BH_4)_2$	6.6	0.066
$Fe(BH_4)_2$	9.4	0.090
$Cr(BH_4)_2$	9.9	0.162
$Mn(BH_4)_2$	9.5	0.174
$Co(BH_4)_2$	9.1	0.264
$Mo(BH_4)_2$	6.4	0.280
$Rh(BH_4)_2$	6.1	0.340
$Ru(BH_4)_2$	6.2	0.351
$Y(BH_4)_3$	9.1	−0.676
$Sc(BH_4)_3$	13.5	−0.595
$Al(BH_4)_3$	16.9	−0.209
$Zr(BH_4)_4$	10.7	−0.429
$Ti(BH_4)_4$	15.0	−0.252

All hydrogen positions were relaxed until the maximum force on the atoms reached 0.05 eV/Å or, alternatively, a maximum of 50 quasi-Newton steps had been performed. The resulting structure was then contracted and expanded to 90%, 95%, 105%, and 110% of the unit cell volume by a proportional scaling of the unit cell, while keeping the B–H distances in each $(BH_4)^-$ group fixed; a single total energy calculation was performed for each volume. A Murnaghan equation-of-state was fitted to the calculated five points to estimate the optimal unit cell volume, to which the unit cell was then scaled (again while conserving B–H distances), followed by a relaxation of the hydrogen positions to a force convergence of 0.05 eV/Å.

After each iteration, an energy versus unit cell volume plot was inspected visually to decide whether the minimum had been sufficiently sampled or an additional iteration of the procedure should be performed; in the latter case, a structure resulting from the first iteration was used as the starting guess for the next iteration.

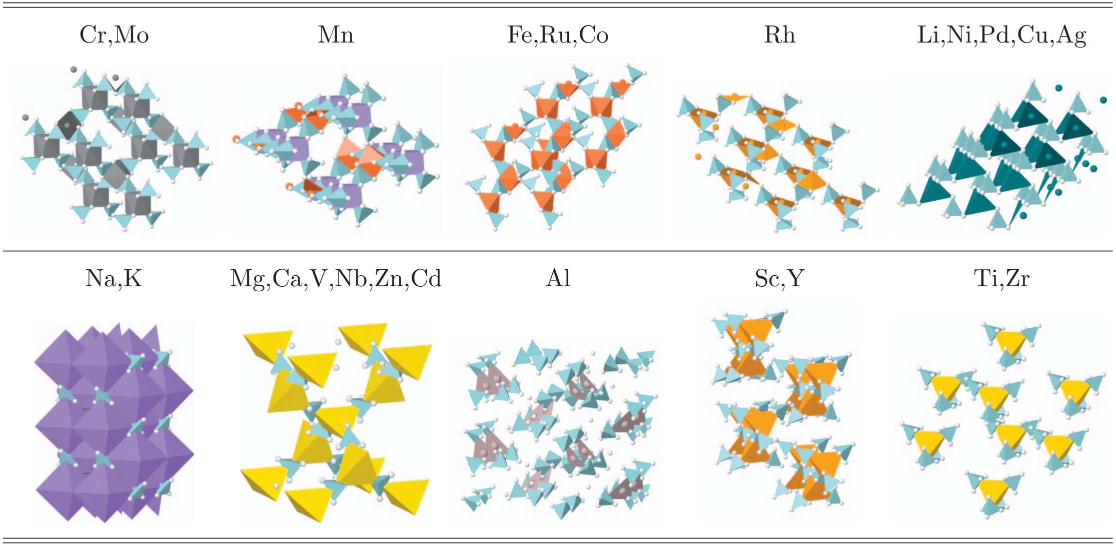


FIG. 2. The structures used for calculating the binary reference energies. For Cr, Mo, Mn, Fe, Ru, Co, Rh, Li, Ni, Pd, Cu, and Ag, the polyhedra show the coordination of the H atoms; the coordination of the $(BH_4)^-$ groups are tetrahedral in these structures. For the remaining metals the coordination polyhedra show the coordination of the $(BH_4)^-$ groups.

2. Step II

When all template structures for each of the (M_1,M_2) alloys had been optimized in step I, the most stable structure was relaxed without constraints by repeating the procedure that first relaxes all atomic positions for a fixed cell and then the unit cell for fixed internal positions. To limit the computational time used by this algorithm, the number of iterations was limited to 5, and the number of steps per iteration was limited to 12 for the internal relaxation and 5 for the unit cell relaxation.

3. Procedure for the additional structures

For the structures calculated later, the $x=2$ structures (monovalent transition metals) followed the same procedure mentioned above, whereas only a single free optimization was performed on the extra $x=3$ and $x=5$ structures, in which all atoms were allowed to relax.

III. DATA COLLECTION AND STORAGE

Every group executed the calculation procedures for steps I and II. After each step, the validity of the results was checked by the group and the results were checked in (stored in a global location for indexing) to the common database.

A. Front end

A Python[20] script took care of checking in all relevant files that were needed for subsequent checking. This included the calculation script and the output files containing the atoms, energies and the calculational parameters. A subversion (svn) version control system[21] assisted to manage groups and users, storing results and assuring transaction consistency.

B. Back end

A second Python script was used to extract the relevant parameters, i.e., the total energy, unit cell volume, chemical symbols, structure, and the calculational parameters such as k-points, number of bands, density wave cutoff, and to select the best structure (at any given time) for every borohydride to create/update the intermediate result plots, which were accessible to all participants. Python, in combination with Matplotlib,[22] was used to ensure a flexible user interface and to generate the plots. A special Python class managed the resulting data, consisting of approximately 5500 calculations. This class provided basic database operations such as selecting, sorting, and filtering of data and facilitated the creation of the plots considerably.

The overall construction of the database and data retrieval procedures will also facilitate screening for possible correlations between combinations of a number of different values in future projects.

IV. DATA ANALYSIS

The initial screening procedure presented here is performed to reduce the number of potential alloys for further investigation, and two simple selection criteria were set up to assess the stability of the investigated alloy structures against phase separation/disproportionation and decomposition. The stabilities were first analyzed against phase separation into the original binary borohydrides as illustrated for $LiSc(BH_4)_4$:

$$\Delta E_{\mathrm{alloy}} = E_{\mathrm{LiSc(BH_4)_4}} - (E_{\mathrm{LiBH_4}} + E_{\mathrm{Sc(BH_4)_3}}). \qquad (1)$$

Reference energies for the 3 alkali, 2 alkaline earth, $Al(BH_4)_3$ plus 19 transition metal borohydrides were ob-

TABLE II. Structures with alloying energies $\Delta E_{\text{alloy}} < 0.0$ eV/f.u. (formula unit) and decomposition energies $\Delta E_{\text{decomp}} < 0.0$ eV/H_2.

	wt % (kg H_2/kg material)	ΔE_{alloy} (eV/f.u.)	ΔE_{decomp} (eV/H_2)
$LiNa(BH_4)_2$	13.5	−0.020	−0.581
$KZn(BH_4)_3$	8.1	−0.349	−0.423
$KAl(BH_4)_4$	12.9	−0.138	−0.416
$NaAl(BH_4)_4$	14.7	−0.279	−0.373
$KCd(BH_4)_3$	6.2	−0.005	−0.352
$NaZn(BH_4)_3$	9.1	−0.358	−0.344
$LiAl(BH_4)_4$	17.3	−0.391	−0.311
$KFe(BH_4)_3$	8.7	−0.116	−0.282
$LiZn(BH_4)_3$	10.4	−0.362	−0.243
$NaFe(BH_4)_3$	9.8	−0.141	−0.206
$KMn(BH_4)_4$	10.5	−0.148	−0.174
$NaNb(BH_4)_4$	9.2	−0.128	−0.165
$KCo(BH_4)_3$	8.5	−0.089	−0.161
$NaMn(BH_4)_4$	11.7	−0.284	−0.131
$KNi(BH_4)_3$	8.5	−0.120	−0.116
$LiFe(BH_4)_3$	11.3	−0.141	−0.104
$LiNb(BH_4)_4$	10.1	−0.194	−0.097
$NaCo(BH_4)_3$	9.6	−0.143	−0.090
$KRh(BH_4)_4$	8.0	−0.058	−0.079
$LiMn(BH_4)_4$	13.3	−0.358	−0.063
$NaNi(BH_4)_3$	9.6	−0.164	−0.043
$NaRh(BH_4)_4$	8.7	−0.033	−0.016

TABLE III. Structures with alloying energies $0 < \Delta E_{\text{alloy}} < 0.2$ eV/f.u. (formula unit) with decomposition energies $\Delta E_{\text{decomp}} < 0.0$ eV/H_2.

	wt % (kg H_2/kg material)	ΔE_{alloy} (eV/f.u.)	ΔE_{decomp} (eV/H_2)
$KNa(BH_4)_2$	8.8	0.095	−0.825
$NaY(BH_4)_4$	9.4	0.115	−0.675
$NaCa(BH_4)_3$	11.2	0.129	−0.645
$LiY(BH_4)_4$	10.4	0.033	−0.609
$LiCa(BH_4)_3$	13.2	0.052	−0.556
$LiSc(BH_4)_4$	14.5	0.143	−0.534
$NaCd(BH_4)_3$	6.7	0.003	−0.271
$KNb(BH_4)_4$	8.4	0.016	−0.207
$NaV(BH_4)_4$	12.1	0.076	−0.188
$NaAg(BH_4)_2$	5.0	0.193	−0.177
$LiCd(BH_4)_3$	7.4	0.102	−0.152
$KCr(BH_4)_4$	10.7	0.199	−0.136
$LiV(BH_4)_4$	13.8	0.061	−0.113
$NaCr(BH_4)_4$	12.0	0.050	−0.095
$KPd(BH_4)_3$	6.4	0.047	−0.095
$KMo(BH_4)_4$	8.3	0.185	−0.079
$KRu(BH_4)_3$	6.5	0.168	−0.061
$NaMo(BH_4)_4$	9.0	0.056	−0.035
$LiCr(BH_4)_4$	13.6	0.029	−0.021
$NaPd(BH_4)_3$	7.0	0.052	−0.014

tained using the most stable structures among the applied $M_2(BH_4)_{1-4}$ model templates (see Table I). Due to computational constraints, the performed calculations are not spin polarized, which causes certain reference structures, e.g., $Mn(BH_4)_2$, to become unstable. In order not to exclude potentially stable candidates, the assessment in Eq. (1) was used for all reference structures (see Table I).

For assessing the stability of alloys with a potentially less favorable stoichiometry, like $LiSc(BH_4)_3$, an effective reference value for $E_{Sc(BH_4)_2}$ was determined from the stable $E_{Sc(BH_4)_3}$ as $E_{Sc(BH_4)_2}{}^* = E_{Sc(BH_4)_3} - 2E_{H_2} - E_B$. Using $1/2(B_2H_6 + H_2)$ as a reference only shifts the energy by 0.07 eV/H_2 and does not result in a new preferred coordination for any of the stable alloys.

The decomposition pathways of binary and ternary metal borohydrides are often highly complex and differ significantly from one system to the next, e.g., $LiBH_4$,[23] $Mg(BH_4)_2$,[24] and $LiZn(BH_4)_3$,[25] and the formed products can even depend on the details of the desorption conditions. Certain compounds form transition metal hydrides,[26] others form transition metal borides,[27] di-,[10] or dodeca-boranes,[28] and others again, e.g., Cr, Cd, Mn, and $Zn(BH_4)_2$ decompose to the elements.[29,30] Given the inclusive nature of this initial screening study and the fact that the true decomposition pathways in most of the investigated alloys are not well known, a simple and generic decomposition pathway was selected, which all interesting mixed borohydrides must be stable against (as a minimum). Here, the alloys decompose into the highly stable alkali- and alkaline earth hydrides, transition metals, boron and H_2, e.g.:

$$\Delta E_{\text{decomp}} = E_{LiMn(BH_4)_3} - (E_{LiH} + E_{Mn} + 3E_B + 5.5E_{H_2}). \tag{2}$$

In this definition, ΔE_{decomp} estimates the stability of the alloy against decomposition. Transition metal hydrides, metal borides, higher order boranates and diborane, which may potentially form, are thus not taken into consideration in this first screening.

The analysis is based on the ground state energies only. Although the difference in vibrational entropy between hydrogen in an alkali metal borohydride and in the gas phase is often significantly smaller than in conventional metal hydrides,[31] the contributions to the free energy from the vibrational entropy may be significant.

A stability range of $\Delta E_{\text{alloy}} \leq 0.0$ eV/f.u. (formula unit) and $\Delta E_{\text{decomp}} \in \{-0.5; 0.0\}$ eV/H_2 is used to select the most interesting alloys with $\Delta E_{\text{decomp}} = -0.2$ eV/H_2 as the target value (see Table II), but given the idealized screening criteria in Eqs. (1) and (2), alloys with only small instabilities, i.e., $\Delta E_{\text{alloy}} \leq 0.2$ eV/f.u and $\Delta E_{\text{decomp}} \leq 0.0$ eV/H_2 should not be discarded *a priori* (see Table III).

V. RESULTS

As the first step of the stability screening, we have plotted the alloying energy against the decomposition energy of the 757 investigated alloys (see Fig. 3). Most of the alloys are found to be stable against decomposition, but the majority are found to be unstable against separation into their binary components ($\Delta E_{\text{alloy}} > 0.0$ eV/f.u.). Many are still within the 0.2 eV/f.u. boundary regime. The lithium-containing alloys (red) are less stable against decomposition than those containing sodium (blue) and potassium (green). Restricting the plot to only the most stable structure for each

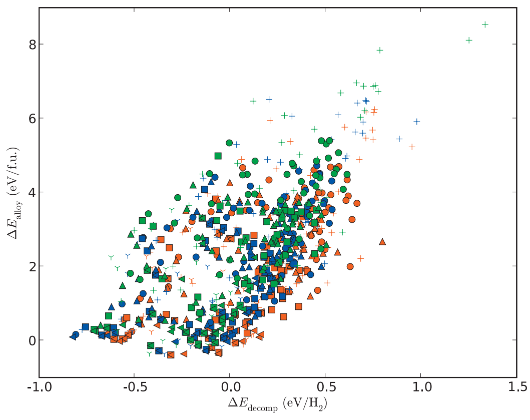


FIG. 3. The alloying energy, ΔE_{alloy}, as a function of the decomposition energy, ΔE_{decomp}, for all alloy compositions. Colors: Li (red), Na (blue), and K (green). Investigated coordinations: tetra (□), octa (○), octa-tetra (△), tetra-octa (+), tetra-tri (ϒ), other (◁). Total number of structures: 757.

M_1M_2 system (see Fig. 4) seems to support this observation, and yields a total of 22 stable alloys (see Table II). Figure 4 is dominated by alloys where (a) both metal atoms are tetrahedrally coordinated to the borohydride groups (□), (b) one is tetrahedral the other trigonal (ϒ), and (c) so-called other (◁), where all constraints have been lifted. Some octa-tetra (△) and tetra-octa (+) are also observed.

Plotting the hydrogen density of the stable alloys, $\Delta E_{alloy} \leq 0.0$ eV/f.u. and $\Delta E_{decomp} \leq 0.0$ eV/H$_2$, Fig. 5 shows that alloys containing potassium (in green) are found to have the lowest density, followed by sodium (in blue) and lithium (in red), as expected. The overall density is found to be around that of liquid hydrogen, which is largely due to the choice of simple template structures; higher densities are expected for real systems as previously observed for $Mg(BH_4)_2$.[9] Alloys containing Al, Mn, Fe, and Zn are found to be stable for all alkali metals screened, whereas those

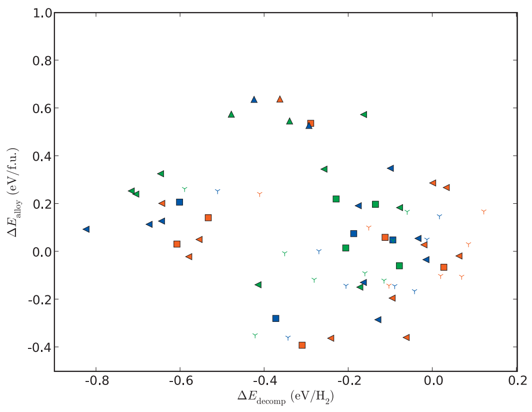


FIG. 4. The alloying energy, ΔE_{alloy}, as a function of the decomposition energy, ΔE_{decomp}, for all preferred alloy systems. Colors: Li (red), Na (blue) and K (green). Preferred local coordination: tetra (□), octa (○), octa-tetra (△), tetra-octa (+), tetra-tri (ϒ), other (◁).

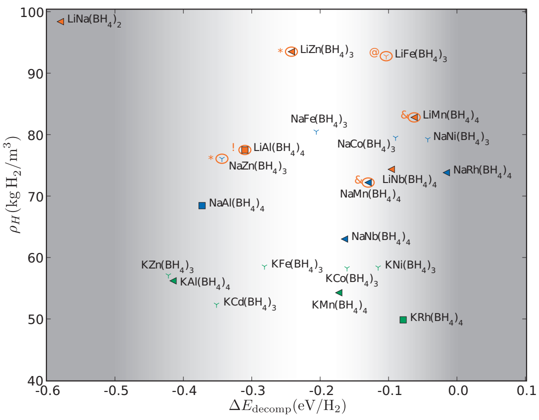


FIG. 5. The hydrogen density (kg H$_2$ m^{-3} as a function of the decomposition energy for the 22 alloys with $\Delta E_{alloy} \leq 0.0$ eV/f.u. and $\Delta E_{decomp} \leq 0.0$ eV/H$_2$. References to experimental observations: !: Ref. 25, @: Ref. 37, &: Ref. 38, *: Ref. 39. Colors: Li (red), Na (blue), and K (green). Preferred local coordination: tetra (□), octa (○), octa-tetra (△), tetra-octa (+), tetra-tri (ϒ), other (◁).

based on Co, Ni, Nb, and Rh are stable for two out of three alkali metals. The only other stable alloys are $KCd(BH_4)_3$ and $LiNa(BH_4)_2$ (see Table II).

The storage capacity (wt % hydrogen) of the stable alloys is plotted as a function of the decomposition energy, ΔE_{decomp}, in Fig. 6. Here, the data from the binary reference structures have also been included, and it is clearly seen that the stability has been reduced significantly compared to the highly stable binary borohydrides. Most alloys have storage capacities above the DOE 2015 system target of 9 wt % (Ref. 3) and several also have favorable stabilities. A number of these ternary borohydrides have been synthesized either

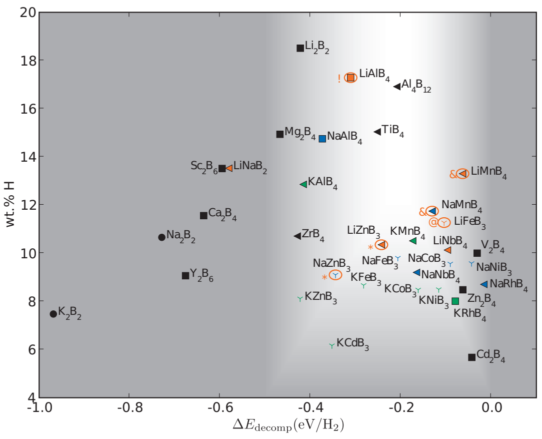


FIG. 6. The weight percent of hydrogen (wt. %) as a function of the decomposition energy, ΔE_{decomp} [Eq. (2)], for all 22 stable alloys and 13 binary reference structures ($\Delta E_{decomp} \leq 0.0$ eV/H$_2$ and $\Delta E_{alloy} \leq 0.0$ eV/f.u.). References to experimental observations: !: Ref. 25, @: Ref. 37, &: Ref. 38, *: Ref. 39. Colors: Li (red), Na (blue), K (green), and reference structures (black). Labels: "MB_x" refers to "$M(BH_4)_x$." Preferred local coordination: tetra (□), octa (○), octa-tetra (△), tetra-octa (+), tetra-tri (ϒ), other (◁).

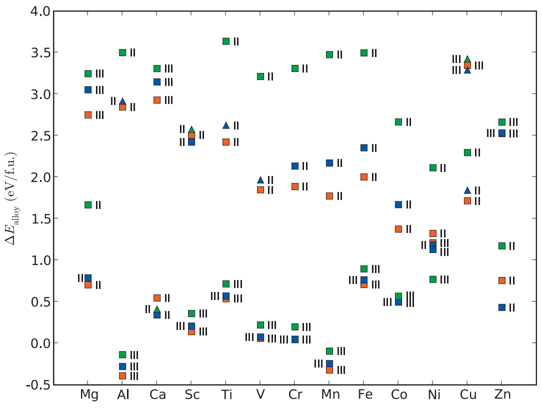


FIG. 7. The alloying energy, ΔE_{alloy}, for the 3*d*-metals (plus Mg, Al, and Ca) in their preferred $M_1M_2(BH_4)_x$ template structures with M_1, M_2, and B fixed for both $x=3$ and $x=4$. Colors: Li (red), Na (blue), and K (green). Preferred local coordination: tetra (□), octa (○), octa-tetra (△), tetra-octa (+). The labels indicate the oxidation state of M_2.

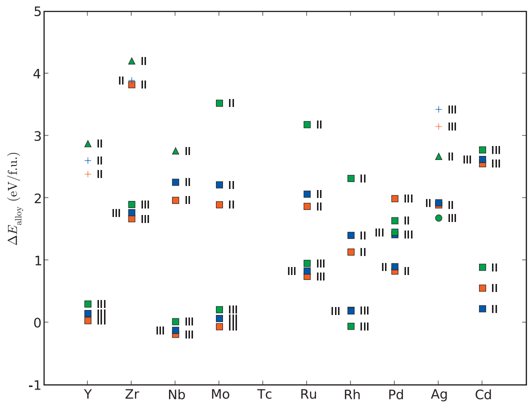


FIG. 8. The alloying energy, ΔE_{alloy}, for the 4*d*-metals in their preferred $M_1M_2(BH_4)_x$ template structures with M_1, M_2, and B fixed for both $x=3$ and 4. Colors: Li (red), Na (blue), and K (green). Preferred local coordination: tetra (□), octa (○), octa-tetra (△), tetra-octa (+). The labels indicate the oxidation state of M_2.

very recently or historically (circled in Figs. 5 and 6). Of the experimentally observed stable/metastable structures, $LiSc(BH_4)_4$,[10] $KNa(BH_4)_2$,[32] and $Li_2Cd(BH_4)_4$ (Ref. 30) show a weak preference for phase separation, but are all found to be potentially stable (see Table III); only $LiK(BH_4)_2$ (Ref. 33) ($\Delta E_{alloy}=0.202$ eV/f.u. and $\Delta E_{decomp}=-0.645$ eV/H_2) and $LiNi(BH_4)_3$ (Ref. 30) ($\Delta E_{alloy}=-0.104$ eV/f.u. and $\Delta E_{decomp}=0.069$ eV/H_2) fall marginally outside the selection criteria. Furthermore, $LiMn(BH_4)_3$ and $NaMn(BH_4)_3$ are found experimentally to decompose at ~100 and 110 °C,[34] and $LiZn(BH_4)_3$ and $LiAl(BH_4)_4$ are found to disproportionate at ~130 °C.[25] These are all structures that are located near the optimal stability in the figure (the nonshaded region).

VI. TRENDS

Given the systematic approach to the screening study it is also possible to extract information from the database about possible trends and correlations, in order to search for predictors and descriptors[35] for the design of future quaternary alloys or alloys with different cation stoichiometries.

A. 3*d* and 4*d* transition metals

The stability of the alloys, as produced by the most stable $x=3$ and $x=4$ initial template structures before the free relaxation, is presented for all 3*d* transition metals (plus Mg, Ca, and Al) in Fig. 7, and for the 4*d* transition metals in Fig. 8. A clear preference for the $M_1M_2(BH_4)_4$-tetra template is observed, which is somewhat surprising, because many of the transition metals have an oxidation state of II in the reference calculations (see Table I). This apparent discrepancy could result from partially non-ionic bonding in these structures, meaning that the coordination of the hydrogen atoms to the metal is the determining factor, not whether the metal has the "correct" valence. For instance, we find no significant energy difference between $Fe_2(BH_4)_3$ and $Fe(BH_4)_2$ as long as the H atoms are octahedrally coordinated to the Fe atom.

Size effects also become apparent here since the $M_1M_2(BH_4)_4$-tetra template is the only template structure that allows the coordination polyhedra of M_1 and M_2 to be relaxed independently. This is supported by the larger spacing between most of the Li, Na, and K alloy energies produced by the other template structures (see Figs. 7 and 8).

To investigate this further, the $M_1M_2(BH_4)_3$-tetra/tri-template was applied to all alloys, and in Figs. 9 and 10, the final alloy stabilities are presented; these also include the free relaxation and the additional $x=2$ and $x=5$ calculations. It is seen that the $M_1M_2(BH_4)_3$-tetra/tri-structures now become the most stable for a number of alloys and that the Li, Na, and K points lie closer indicating a reduction in the size effects.

There is a general agreement between valencies in the reference calculations and the alloys; divalent metals are

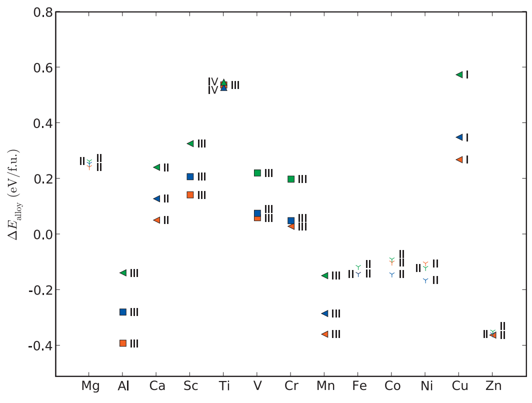


FIG. 9. The alloying energy, ΔE_{alloy}, for the 3*d*-metals using only the energy of the preferred $M_1M_2(BH_4)_x$, $x=2-5$ structure. Colors: Li (red), Na (blue), and K (green). Preferred local coordination: tetra (□), octa (○), octa-tetra (△), tetra-octa (+), tetra-tri (ᛉ), other (◁). The labels indicate the oxidation state of M_2.

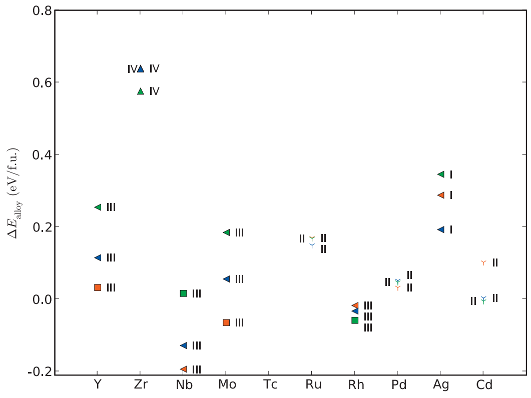


FIG. 10. The alloying energy, $\Delta E_{\mathrm{alloy}}$, for the 4*d*-metals using only the energy of the preferred $M_1M_2(\mathrm{BH}_4)_x$, $x=2-5$ structure. Colors: Li (red), Na (blue), and K (green). Preferred local coordination: tetra (□), octa (○), octa-tetra (△), tetra-octa (+), tetra-tri (ᛉ), other (◁). The labels indicate the oxidation state of M_2.

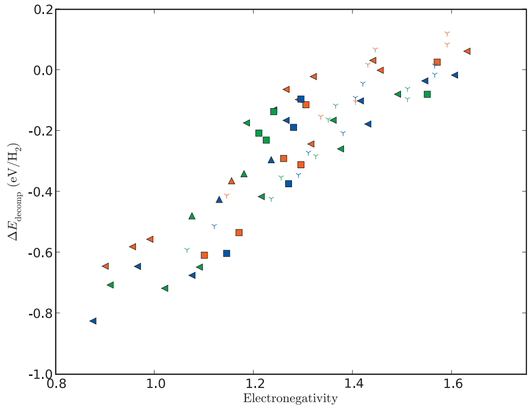


FIG. 11. The decomposition energy, $\Delta E_{\mathrm{decomp}}$, as a function of the average Pauling electronegativity for all alloys in their preferred $M_1M_2(\mathrm{BH}_4)_x$ coordination: tetra (□), octa (○), octa-tetra (△), tetra-octa (+), tetra-tri (ᛉ), other (◁). Colors: Li (red), Na (blue), and K (green).

found to prefer a $M_1M_2(\mathrm{BH}_4)_3$ configuration, whereas trivalent metals prefer $M_1M_2(\mathrm{BH}_4)_4$, tetravalent metals prefer $M_1M_2(\mathrm{BH}_4)_5$ and the monovalent Cu and Ag prefer $M_1M_2(\mathrm{BH}_4)_2$. Some deviations are found, but given the simple model structures used for both alloys and reference calculations, and given the fact that some of the metals are found by experiments to form ternary borohydrides in different oxidation states, the agreement is good.

The most stable alloys are found for the half-filled *d*-bands, but interesting alloys are also found for the empty and fully occupied *d*-bands with the addition of Al, where the $M_1\mathrm{Al}(\mathrm{BH}_4)_4$ are found to be promising (see Figs. 9 and 10).

Lithium-based alloys (red) are generally found to be the most stable, followed by sodium (blue) and potassium (green), although significant deviations are observed. This follows the observed trend for the storage capacities.

B. Stability versus electronegativity

A number of recent publications[33,36] have shown an apparent linear correlation between the decomposition temperature and the average cation Pauling electronegativity. Although this might be expected, given the definition of Pauling's electronegativity, it also indicates that the kinetic barriers—if any—do not appear to be particularly system dependent.

Plotting the calculated decomposition energy as a function of the average cation electronegativity for all alloys in their most stable local coordination (see Fig. 11) appears to support this observation. The scatter of the data points around the "line" (which would have a slope that agrees with Ref. 36 to within 10%–15%) is, however, significant and deviations of ± 0.1 eV/H$_2$ can be sufficient to shift a material from interesting to irrelevant for storage applications, or vice versa.

The stable alloys ($\Delta E_{\mathrm{alloy}} \leq 0.0$ eV/f.u.) are seen to cluster around certain average electronegativities of 1.3–1.4 and 1.6 (see Fig. 12). The cluster around 1.3–1.4 is highly promising with $\Delta E_{\mathrm{decomp}} \simeq -0.1$ eV/H$_2$ for Mn and Nb and particularly promising for Al, Zn, and Fe with $\Delta E_{\mathrm{decomp}} \simeq -0.3$ eV/H$_2$. The Mo and Rh alloys at electronegativities around 1.6 are found to border on decomposition, but experimental work by Nikels *et al.*[33] estimates the decomposition temperature of such compounds to be around 150 °C.

VII. CONCLUSIONS

We have analyzed the thermodynamic properties of possible alkali-transition metal borohydride systems, finding a number of candidates showing favorable properties.

The $M_1(\mathrm{Al/Mn/Fe})(\mathrm{BH}_4)_4$, $(\mathrm{Li/Na})\mathrm{Zn}(\mathrm{BH}_4)_3$, and $(\mathrm{Na/K})(\mathrm{Ni/Co})(\mathrm{BH}_4)_3$ alloys are found to be the most promising, followed by selected $M_1(\mathrm{Nb/Rh})(\mathrm{BH}_4)_4$ alloys. These findings are in good agreement with experimental observations for $\mathrm{LiFe(BH)}_3$,[37] $\mathrm{LiAl}(\mathrm{BH}_4)_4$,[25] $(\mathrm{Li/Na})\mathrm{Mn(BH)}_{3,4}$,[38]

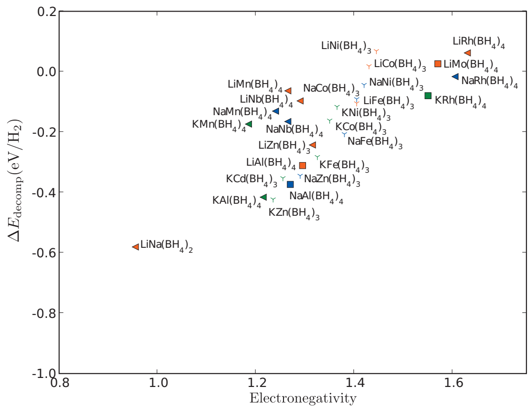


FIG. 12. The decomposition energy, $\Delta E_{\mathrm{decomp}}$, as a function of the average Pauling electronegativity for alloys with $\Delta E_{\mathrm{alloy}} \leq 0.0$ eV/H$_2$. Colors: Li (red), Na (blue), and K (green). Preferred $M_1M_2(\mathrm{BH}_4)_x$, $x=2-5$, coordination: tetra (■), octa (○), octa-tetra (△), tetra-octa (+), tetra-tri (ᛉ), other (◁).

and (Li/Na)Zn$(BH_4)_3$,[39] whereas the Co, Cd, Nb, and Rh and alloys still remain to be synthesized and tested. Although some structures can be observed experimentally in different metal-metal stoichiometries than those used in the screening study, e.g., the Li–Zn system,[39] the alloy systems were still identified as promising candidates in this screening study. Some of the nearly stable compounds in Table III, e.g., LiSc$(BH_4)_4$ (Ref. 10) and KNa$(BH_4)_2$ (Ref. 32) have recently been found to be metastable, while LiNi$(BH_4)_3$ (Ref. 30) was found to be marginally unstable here. The LCS approach was found to limit the 757 potential alloys to 22 promising candidates of which ~10 are highly promising. These structures can now be pursued further, analyzing their detailed decomposition pathways, both theoretically[40] and experimentally.

ACKNOWLEDGMENTS

The authors acknowledge financial support by the European Commission DG Research (Contract No. SES6-2006-51827/NESSHy), the Nordic Energy Research Council (Contract No. 06-HYDRO-C15) and the Danish Center for Scientific Computing (DCSC) for computer time (Grant No. HDW-1103-06). The Center for Atomic-Scale Materials Design is funded by the Lundbeck Foundation.

[1] L. Schlapbach and A. Züttel, Nature **414**, 353 (2001).
[2] D. Mosher, X. Tang, and S. Arsenault, DoE Hydrogen Program, FY 2006 Annual Progress Report, 2006, pp. 281–284.
[3] See http://www1.eere.energy.gov/vehiclesandfuels/about/partnerships/freedomcar/fc_goals.html for FreedomCAR and Fuel Technical Partnership Technical Goals.
[4] S. Orimo, Y. Nakamori, J. R. Eliseo, A. Züttel, and C. M. Jensen, Chem. Rev. **107**, 4111 (2007).
[5] R. Z. Sørensen, J. S. Hummelshøj, A. Klerke, J. B. Reves, T. Vegge, J. K. Nørskov, and C. H. Christensen, J. Am. Chem. Soc. **130**, 8660 (2008).
[6] A. Züttel, S. Rentsch, P. Fischer, P. Wenger, P. Sudan, P. Mauron, and C. Emmenegger, J. Alloys Compd. **356–357**, 515 (2003).
[7] Z. Łodziana and T. Vegge, Phys. Rev. Lett. **93**, 145501 (2004).
[8] Z. Łodziana and T. Vegge, Phys. Rev. Lett. **97**, 119602 (2006).
[9] K. Chłopek, C. Frommen, A. Léon, O. Zabara, and M. Fichtner, J. Mater. Chem. **17**, 3496 (2007).
[10] H. Hagemann, M. Longhini, J. W. Kaminski, T. A. Wesolowski, R. Cerný, N. Penin, M. H. Sørby, B. C. Hauback, G. Severa, and C. M. Jensen, J. Phys. Chem. A **112**, 7551 (2008).
[11] S. V. Alapati, J. K. Johnson, and D. S. Sholl, J. Phys. Chem. C **112**, 5258 (2008).
[12] V. Ozolins, E. H. Majzoub, and C. Wolverton, Phys. Rev. Lett. **100**, 135501 (2008).
[13] J. Voss, J. S. Hummelshøj, Z. Łodziana, and T. Vegge, J. Phys.: Condens. Matter **21**, 012203 (2009).
[14] P. Hohenberg and W. Kohn, Phys. Rev. **136**, B864 (1964).
[15] B. Hammer, L. B. Hansen, and J. K. Nørskov, Phys. Rev. B **59**, 7413 (1999).
[16] D. Vanderbilt, Phys. Rev. B **41**, 7892 (1990).
[17] S. R. Bahn and K. W. Jacobsen, Comput. Sci. Eng. **4**, 56 (2002).
[18] D. F. Shanno, Math. Comput. **24**, 647 (1970).
[19] F. Buchter, Z. Łodziana, A. Remhof, O. Friedrichs, A. Borgschulte, Ph. Mauron, A. Züttel, D. Sheptyakov, G. Barkhordarian, R. Bormann, K. Chłopek, M. Fichtner, M. Sørby, M. Riktor, B. Hauback, and S. Orimo, J. Phys. Chem. B **112**, 8042 (2008).
[20] See http://www.python.org for Python Programming Language.
[21] See http://subversion.tigris.org for Subversion version control system.
[22] See http://matplotlib.sourceforge.net for Matplotlib 2D plotting library for Python.
[23] J.-H. Her, M. Yousufuddin, W. Zhou, S. S. Jalisatgi, J. G. Kulleck, J. A. Zan, S.-J. Hwang, R. C. Bowman, Jr., and T. J. Udovic, Inorg. Chem. **47**, 9757 (2008).
[24] N. Hanada, K. Chłopek, C. Frommen, W. Lohstroh, and M. Fichtner, J. Mater. Chem. **18**, 2611 (2008).
[25] H.-W. Li, S. Orimo, Y. Nakamori, K. Miwa, N. Ohba, S. Towata, and A. Züttel, J. Alloys Compd. **446–447**, 315 (2007).
[26] E. Wiberg, Angew. Chem. **65**, 16 (1953).
[27] X. B. Yu, D. M. Grant, and G. S. Walker, Chem. Commun. (Cambridge) **37**, 3906 (2006).
[28] S.-J. Hwang, R. C. Bowman, Jr., J. W. Reiter, J. Rijssenbeek, G. L. Soloveichik, J.-C. Zhao, H. Kabbour, and C. C. Ahn, J. Phys. Chem. C **112**, 3164 (2008).
[29] Y. Nakamori, H.-W. Li, M. Matsuo, K. Miwa, S. Towata, and S. Orimo, J. Phys. Chem. Solids **69**, 2292 (2008).
[30] Y. Nakamori and S. Orimo, in *Solid-state Hydrogen Storage—Materials and Chemistry*, edited by G. Walker (Woodhead, Cambridge, 2008), pp. 420–449.
[31] P. Mauron, F. Buchter, O. Friedrichs, A. Remhof, M. Bielmann, C. N. Zwicky, and A. Züttel, J. Phys. Chem. B **112**, 906 (2008).
[32] L. Seballos, J. Z. Zhang, E. Ronnebro, J. L. Herberg, and E. H. Majzoub, J. Alloys Compd. **476**, 446 (2009).
[33] E. A. Nickels, M. O. Jones, W. I. F. David, S. R. Johnson, R. L. Lowton, M. Sommariva, and P. P. Edwards, Angew. Chem., Int. Ed. **47**, 2817 (2008).
[34] See http://www.docstoc.com/docs/922287/Fundamental-Studies-of-Advanced-High-Capacity-Reversible-Metal-Hydrides for "Fundamental Studies of Advanced High-Capacity, Reversible Metal Hydrides," presentation by C. M. Jensen, DOE Hydrogen Program, 13 May 2007.
[35] F. Studt, F. Abild-Pedersen, T. Bligaard, R. Z. Sørensen, C. H. Christensen, and J. K. Nørskov, Science **320**, 1320 (2008).
[36] Y. Nakamori, K. Miwa, A. Ninomiya, H. Li, N. Ohba, S. Towata, A. Züttel, and S. Orimo, Phys. Rev. B **74**, 045126 (2006).
[37] H. Nöth and P. Fritz, Angew. Chem. **73**, 408 (1961).
[38] See http://www.hydrogen.energy.gov/pdfs/review08/st_0_satyapal.pdf for "Hydrogen Storage," presentation by S. Satyapal, 2008 DOE Hydrogen Program, Merit Review and Peer Evaluation Meeting, 9 June 2008.
[39] H. Nöth, E. Wiberg, and L. Winter, Z. Anorg. Allg. Chem. **386**, 73 (1971).
[40] V. Ozolins, E. H. Majzoub, and C. Wolverton, J. Am. Chem. Soc. **131**, 230 (2009).
[41] See EPAPS Document No. E-JCPSA6-130-043923 for a listing of the authors' affiliations. For more information on EPAPS, see http://www.aip.org/pubservs/epaps.html.

Appendix D

Virtual Materials Design using Databases of Calculated Materials Properties

T. R. Munter, D. D. Landis,F. Abild Pedersen, G. Jones, S. Wang and T. Bligaard
Computational Science & Discovery 2 (2009) 015006 (27pp)

COMPUTATIONAL SCIENCE&DISCOVERY

Virtual materials design using databases of calculated materials properties

T R Munter[1], D D Landis[1], F Abild-Pedersen[1,2], G Jones[1], S Wang[1] and T Bligaard[1]

[1] Center for Atomic-scale Materials Design (CAMD), Department of Physics, Technical University of Denmark, Building 307, DK-2800 Kgs. Lyngby, Denmark
[2] Computational Materials Design ApS, Fysikvej Building 307, DK-2800 Kgs. Lyngby, Denmark
E-mail: bligaard@fysik.dtu.dk

Received 27 December 2007, in final form 28 October 2009
Published 26 November 2009
Computational Science & Discovery **2** (2009) 015006 (27pp)
doi:10.1088/1749-4699/2/1/015006

Abstract. Materials design is most commonly carried out by experimental trial and error techniques. Current trends indicate that the increased complexity of newly developed materials, the exponential growth of the available computational power, and the constantly improving algorithms for solving the electronic structure problem, will continue to increase the relative importance of computational methods in the design of new materials. One possibility for utilizing electronic structure theory in the design of new materials is to create large databases of materials properties, and subsequently screen these for new potential candidates satisfying given design criteria. We utilize a database of more than 81 000 electronic structure calculations. This alloy database is combined with other published materials properties to form the foundation of a virtual materials design framework (VMDF). The VMDF offers a flexible collection of materials databases, filters, analysis tools and visualization methods, which are particularly useful in the design of new functional materials and surface structures. The applicability of the VMDF is illustrated by two examples. One is the determination of the Pareto-optimal set of binary alloy methanation catalysts with respect to catalytic activity and alloy stability; the other is the search for new alloy mercury absorbers.

Contents

1. Introduction

During the last century, materials science was subject to an enormous transformation. Based on the development of the quantum theory of physics, the human control over matter saw a number of successes leading up to the materials technology of today, where the human genome has been mapped [1] and measurements with, e.g., scanning probe microscopes and transmission electron microscopes are used to create images of surfaces with sub-nanometer resolution [2, 3]. The subsequent influence on the development of new materials is dramatic. One area where the development of simple models from quantum mechanical principles has had a profound effect is organic synthesis, where chemists today almost routinely create molecules with the exact atomic configurations they seek.

However, there are still many problems facing the materials scientists who are attempting to design new materials with predefined design objectives. One fundamental problem is the large number of different possible atomic arrangements that can be envisioned for even a very limited number of atoms. This leads to an intractable number of design possibilities for materials, each with significantly varying properties, and among these perhaps only a few have the desired properties or are practically realizable. Despite the cost of discovering new materials with specific properties continuously decreasing due to the introduction of high-throughput parallel synthesis and screening techniques, the cost for developing new materials is still high.

The modern implementations of electronic structure theory provide hope that better solutions to this problem are getting closer. Methodological improvements continuously appear, which increase the accuracy as well as the speed of the computation of materials properties. The cost for computational power is in parallel to the methodological improvements decreasing exponentially, and has been so since perhaps the early 1960s following Moore's law (stating that a doubling in the number of transistors that can be integrated on a single circuit can be expected every two years) [4]. This has led to the situation that we today, in some fortunate cases, can perform computer simulations to obtain the relevant materials properties faster, more reliably, and at lower cost than the traditional synthesis and testing procedure. As an example of this, we shall in this paper use a database of more than 81 000 density functional theory (DFT) calculations for various materials and properties to screen for stable methanation catalysts and mercury absorbers. An accurate experimental determination of the materials properties for such a large number of compounds might be difficult to obtain in a lifetime of work effort. The computational work load, however, is large but not excessive.

Extrapolating the advances of the recent past suggests that electronic structure theory might in the relatively near future become a highly competitive tool in the materials design process for a much wider range of materials.

When intending to calculate materials properties or mechanisms for chemical reactions at the atomic scale, resolving the electronic structure is necessary. Presently, DFT [5–7] is the method of choice for calculating electronic structure to a reasonably high degree of accuracy with reasonably limited computational effort, when the system size is from a few atoms up to a few hundred atoms. DFT is versatile in the sense that by solving the electronic structure problem using DFT, one can determine a wide range of atomic-scale properties for a given atomic-scale structure. One can even predict the properties for materials which have not yet been synthesized [8, 9], or for which the relevant synthesis and testing might be too expensive or even dangerous. The otherwise clear goal of predicting new materials with improved properties based on computational tools immediately raises a series of important questions, which do not have any obvious answers: how does one obtain the materials properties, which are observed at larger length and time scales from the calculations at the atomic scale? Is it already possible today to establish large databases of materials properties calculated by these theoretical tools? How do we efficiently use such databases for materials optimization when multiple criteria need to be optimized simultaneously?

These questions are perhaps too difficult to answer in general within just one short paper. However, an attempt will be made here to create the foundation on which the detailed answers for these questions in specific cases can subsequently be developed. The focus throughout the paper will be on catalysis and ordered intermetallic alloys and on the absorption of heavy metals in alloys. We discuss a practical implementation of the data analysis with the purpose of facilitating various aspects of the prediction of materials properties and chemical reactions using calculated materials properties.

Within the pharmaceutical industry *virtual screening* methods have for a long time been used for drug discovery. There, large libraries of chemical compounds or biological molecules are evaluated automatically for desired functionality using computers. For drug discovery virtual screening is used for estimating among others: the absorption of drugs in the body, metabolism, toxicity and evaluation of drug activity of chemical compounds [10–12]. Virtual screening is a relatively new concept in the heterogeneous catalysis field [13]. Whereas medical companies have big databases of well-characterized compounds and their properties, it has proven more difficult to determine the active surface sites at the atomic scale under conditions that are relevant to catalytic processes. This has in turn slowed down the progress of high-throughput theoretical design of new catalysts. Computational methods have been useful for understanding trends in the properties of catalysts across the Periodic Table, and have been used widely within the field of theoretical catalysis. The atomic-level understanding of ammonia synthesis [14–16], methanation [17], ethylene epoxidation [18], the oxygen reduction reaction [19], CO oxidation [20, 21], the hydrogen evolution reaction [22] and methanol decomposition [23] among others was facilitated by the use of computational methods.

Progress has recently been made with respect to actual theoretical design [24–36]. Direct atomic-scale computational design is difficult due to the lack of databases with chemical properties relevant for catalyst development.

In this paper, we present the implementation of a program tailored to carrying out virtual screening for the design of catalysts for heterogeneous and electrochemical processes. The implemented approach to virtual screening is an example of what has previously been called *data pipelining* and been used for drug discovery [37], and is simultaneously inspired by the ‘Cambridge Engineering Selector’ and related works by Professor M F Ashby [38].

Catalysis of commodity chemicals is economically important for society. It has been estimated that 20% of the World’s GNP is directly related to catalytic processes [39]. In this paper, we therefore use catalysis as an example, although the virtual materials design framework (VMDF) we present is more generally applicable for computational materials design.

2. Overview

The virtual screening program VMDF that we present is implemented in Java, and can therefore run on most hardware platforms. The program provides easy access to an extensive embedded database of computationally obtained physical and chemical properties for bulk materials and interaction of molecules with surfaces relevant in catalysis and electrochemistry. The filters and analysis tools, which are provided have been

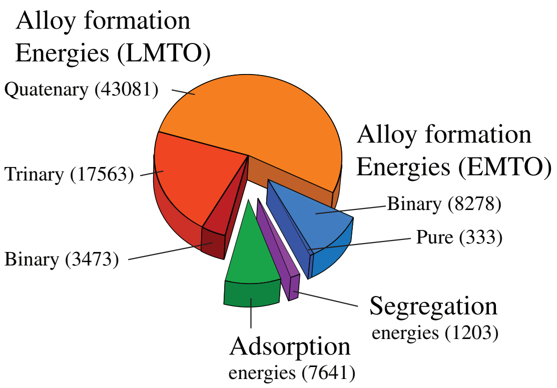


Figure 1. Schematic overview over the sizes of the databases produced at the Center for Atomic-scale Materials Design (CAMD) at the Technical University of Denmark over the past 20 years. The data sets represent theoretical data such as formation energies and segregation energies for bulk alloy systems, adsorption energies and theoretical relations for many catalytically relevant quantities.

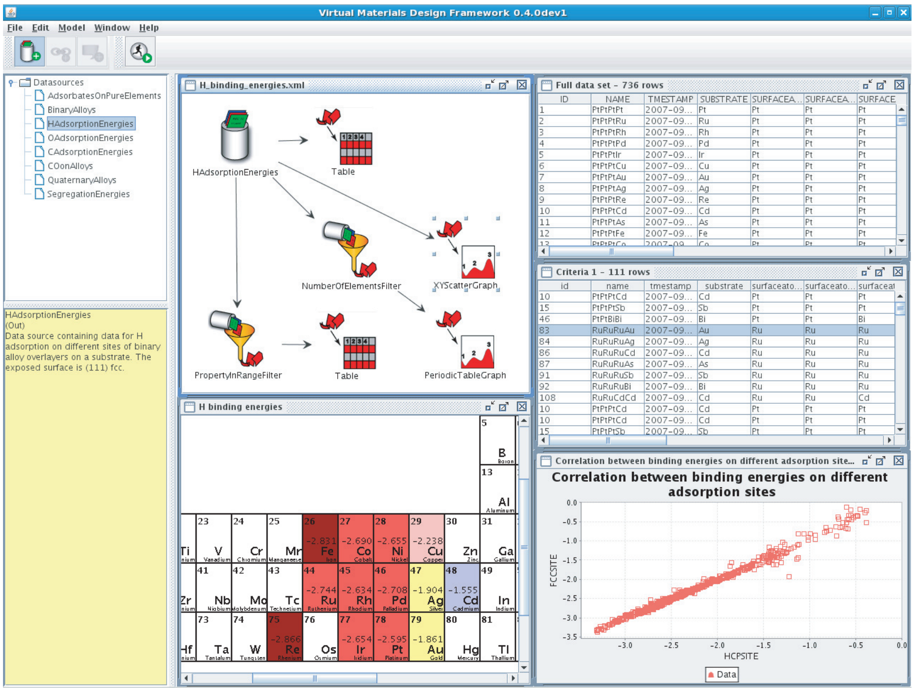
developed to simplify the process of searching and correlating data from multiple databases, in order to find the best leads for new catalysts.

Currently, the database contains electronic structure calculations for more than 81 500 atomic-scale structures. Figure 1 shows in schematic form the size of each part of the database. As computers become increasingly powerful, the process of generating data of chemical properties for adsorbates and surfaces becomes faster. In the near future, we will thus see increasingly large data sets published for increasingly more complex materials properties.

A graphical user-interface (GUI) is used to create a flowchart for the virtual screening. Data sets flow from *data sources*, through *filters* to *visualization tools*, which can present the resulting data sets as, for example, graphs or show them in spreadsheets.

Figure 2 shows a typical screenshot from a virtual screening. To the left is a list of the available filters and a button to show the visualization tools. The model-view window contains a flow-chart model containing one data source, two filters and four visualization tools. The data set containing adsorption energies of hydrogen on binary surface alloys [33], is shown in tables both before and after it was filtered through the `PropertyInRangeFilter`. This filter can be used to selectively allow data elements to pass for which a chosen property has a numerical value in the predefined range. The two corresponding tables can be seen in the upper-right and middle-right windows. The data set is also piped through the `NumberOfElementsFilter` which is set to only let adsorption energies pass for surfaces of pure elements. The data are then plotted in a color-coded Periodic Table, see the lower-middle window. The data set is also piped to a visualization tool that plots the data in a graph. Here the binding energy of H sitting in a face-centered cubic (fcc)-like three-fold adsorption site on the (111) surface of the alloys having a fcc-like crystal structure, is shown as a function of the binding energy of H sitting in a hexagonal close-packed (hcp)-like three-fold adsorption site on the same surface. As one would expect there is a clear linear correlation between the adsorption energy of H in the two adsorption sites.

In section 3 of this paper, the data sets included in our database are presented. In section 4, the underlying architecture of the virtual screening program is described. Section 5 gives an example of how virtual screening can be used to search for metals that can potentially absorb mercury from a gas stream, and bind the mercury in an alloy. Finally, in section 6 we present an example of how physical knowledge about a reaction can be used together with virtual screening to find new leads in the search for a catalyst suitable for the classical methanation reaction.



HAdsorptionEnergies
(Out)
Data source containing data for H adsorption on different sites of binary alloy overlayers on a substrate. The exposed surface is (111) fcc.

Full data set – 736 rows

ID	NAME	TMESTAMP	SUBSTRATE	SURFACEA...	SURFACEA...	SURFACE
1	PtPtPtPt	2007-09...	Pt	Pt	Pt	Pt
2	PtPtPtRu	2007-09...	Ru	Pt	Pt	Pt
3	PtPtPtRh	2007-09...	Rh	Pt	Pt	Pt
4	PtPtPtPd	2007-09...	Pd	Pt	Pt	Pt
5	PtPtPtIr	2007-09...	Ir	Pt	Pt	Pt
6	PtPtPtCu	2007-09...	Cu	Pt	Pt	Pt
7	PtPtPtAu	2007-09...	Au	Pt	Pt	Pt
8	PtPtPtAg	2007-09...	Ag	Pt	Pt	Pt
9	PtPtPtRe	2007-09...	Re	Pt	Pt	Pt
10	PtPtPtCd	2007-09...	Cd	Pt	Pt	Pt
11	PtPtPtAs	2007-09...	As	Pt	Pt	Pt
12	PtPtPtFe	2007-09...	Fe	Pt	Pt	Pt
13	PtPtPtCo	2007-09	Co	Pt	Pt	Pt

Criteria 1 – 111 rows

id	name	tmestamp	substrate	surfaceato...	surfaceato...	surfaceat
10	PtPtPtCd	2007-09...	Cd	Pt	Pt	Pt
15	PtPtPtSb	2007-09...	Sb	Pt	Pt	Pt
46	PtPtBiBi	2007-09...	Bi	Pt	Pt	Bi
83	RuRuRuAu	2007-09...	Au	Ru	Ru	Ru
84	RuRuRuAg	2007-09...	Ag	Ru	Ru	Ru
86	RuRuRuCd	2007-09...	Cd	Ru	Ru	Ru
87	RuRuRuAs	2007-09...	As	Ru	Ru	Ru
91	RuRuRuSb	2007-09...	Sb	Ru	Ru	Ru
92	RuRuRuBi	2007-09...	Bi	Ru	Ru	Ru
108	RuRuCdCd	2007-09...	Cd	Ru	Ru	Cd
10	PtPtPtCd	2007-09...	Cd	Pt	Pt	Pt
10	PtPtPtCd	2007-09...	Cd	Pt	Pt	Pt
15	PtPtPtSb	2007-09...	Sb	Pt	Pt	Pt

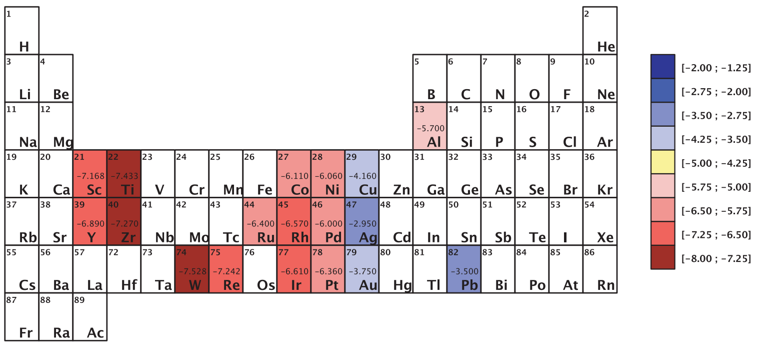
Figure 2. Screenshot of VMDF showing a more advanced model containing one data source, two filters and four visualization tools. Four output windows are shown. The scatter graph (bottom right) shows the adsorption energy of H on two different threefold adsorption sites on binary alloys as a function of each other.

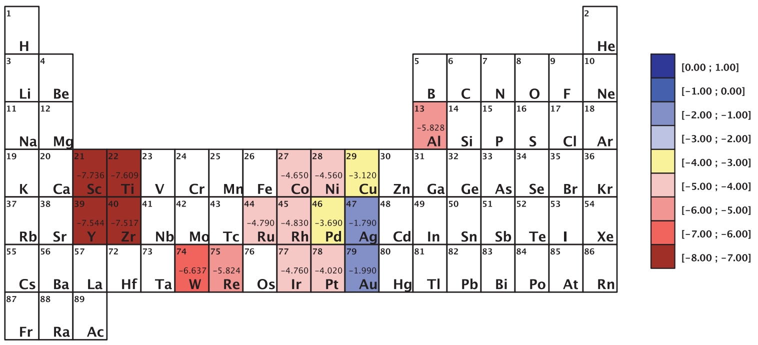
3. A portfolio of databases

Since the introduction of computers in theoretical physics and chemistry, vast amounts of data have been generated. These are often calculated using varying accuracy and are stored in a format suitable for the specific project they were generated for. This makes it a nontrivial task to directly combine results from multiple sources and use them in subsequent projects.

In this paper, we describe one way that data from different sources can be put in a homogeneous form so that different studies can be more easily combined and used to conduct a screening study. One difficulty is often that data from different studies must be put in a consistent form having, for example, the same reference values. Evidently, it would be a more satisfactory solution if one could build the database up on total energies, since the addition of extra materials properties might be simpler. One would in this case, however, need to take much greater care that the reference energies produced and included in the past were of the same computational quality and level as the subsequently included total energies. In the studies presented here, we do not propose a general solution for the problem of mutual reference values. Rather we use a number of data sets that have already been published separately. This means, for example, that whereas the calculations of adsorption energies and alloy formation energies were done with different electronic structure codes, the full calculation of each property is done using energy differences within only one code at one consistent level of accuracy.

Over the past 20 years, the research group which we are part of, at the Technical University of Denmark, has carried out numerous theoretical studies, thereby generating vast amounts of data. Especially, in the last few years the amount of theoretically calculated accurate materials properties has increased tremendously. Among the systems studied are ammonia synthesis [14, 15], the methanation reaction [31], adsorption energies for molecules on transition metal surfaces [17, 40, 41], activation energies for many molecules on transition metal





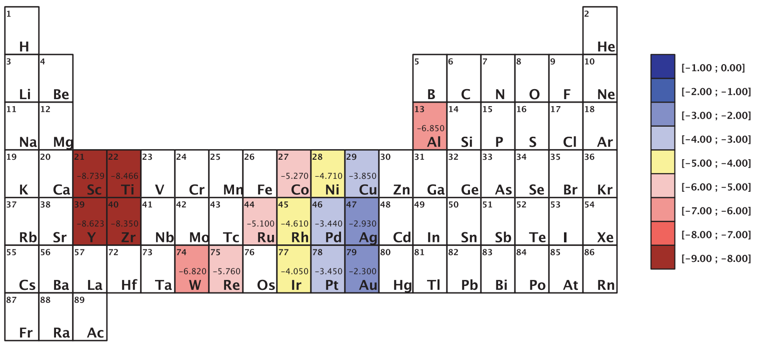


Figure 3. The CAMD database contains many theoretically calculated adsorption energies of adsorbates on the close-packed (111) surface of pure elements. Here is shown a small subset. (a) Adsorption energies for carbon. (b) Adsorption energies for nitrogen. (c) Adsorption energies for oxygen.

surfaces [42] and compounds [43], and the formation energy of quaternary alloys having body-centered cubic (bcc)- or fcc-like crystal structures [26, 44]. Bringing these data into a homogeneous form that can be used in conjunction provides a versatile database.

A fundamental chemical property in catalysis is the energy change when a molecule adsorbs on a surface, the adsorption energy. Figure 3 presents theoretical adsorption energies for carbon-, nitrogen- and oxygen atoms adsorbed on close-packed metal surfaces obtained using DFT. Calculated adsorption energies as presented here can be used for predicting chemical properties for many different alloys. This is sometimes

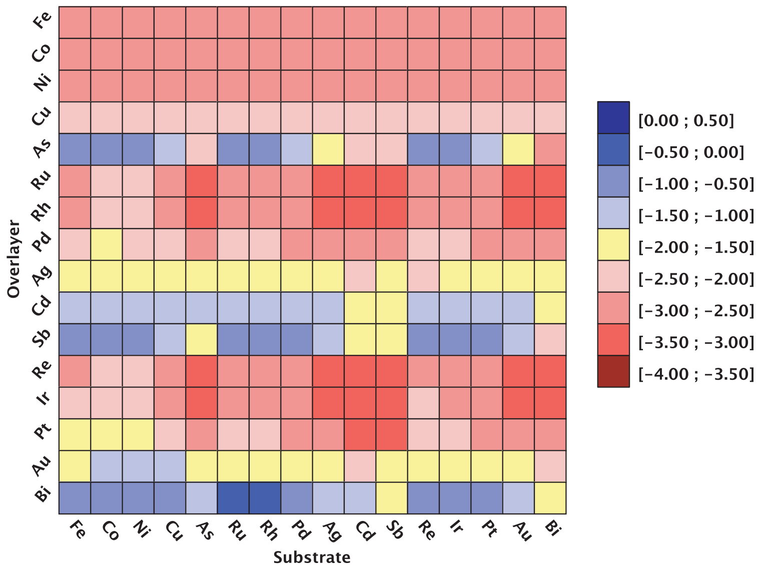


Figure 4. The adsorption energy of hydrogen on a supported overlayer. Horizontally, the element of the substrate is shown. Vertically, the nature of the overlayer is shown. The data are adapted from [33].

facilitated by means of simple interpolation or more advanced schemes that generate approximate materials data, which can subsequently be screened for specific properties. This approach was used successfully when proposing new catalysts for, e.g., the methanation reaction [31].

Chemical properties like adsorption energies can also be tuned by forming a monolayer of one element or a multi-component alloy on a substrate of another metal [45–47]. The change in adsorption energy can primarily be attributed to two effects that govern the position of the material's d-band. The first could be considered an intrinsic property of the chemical elements present and is a result of the different number of valence electrons. The second is a geometric influence brought about by a mismatch in lattice constants of the overlayer and the substrate, which for transition metals changes the width and the position of the d-band [48]. How the adsorption energy of atomic H changes when adsorbed on a surface with a one atomic-layer thick overlayer of one element on a substrate of another element can be seen in figure 4. The diagonal elements in figure 4 provide the adsorption energies of H on the surface of the pure elements.

In earlier works, properties such as adsorption energies, barriers to dissociate and other relevant properties were only calculated for adsorbates on the surfaces of metals relevant to the chemical reaction in question. So when data sets are collected the data for adsorbates will be incomplete with respect to the elements of the surfaces. Figure 5 shows a histogram of the number of different adsorbates for which there exists adsorption energies on a given transition metal surface. Here selected adsorption data on close-packed surfaces which for metals having the fcc crystal structure (the (111)-surface) are shown.

In the present study of methanation catalysts, we used a database of DFT calculations in the full charge density EMTO implementation [49–52] of formation energies for 8611 binary alloys [53]. These binary alloys are those binary combinations of Li, Na, Be, Mg, Ca, Sr, Al, Zn, Ge, Si and the 3d, 4d and 5d transition metals (except La) which sit in the structures shown in figure 6. It was not possible to converge every single combination, since the implementation is not fully automated, but by far the most combinations have been calculated. It will indeed most often be the case, that when working with theoretically calculated materials properties, the databases are not necessarily complete, and subsequent screening studies will be carried out utilizing incomplete data sets. An example of the contents (data for one crystal structure) of this database is shown in figure 7. Here the binary alloys in the $L1_2$ structure are shown with a color-code illustrating their

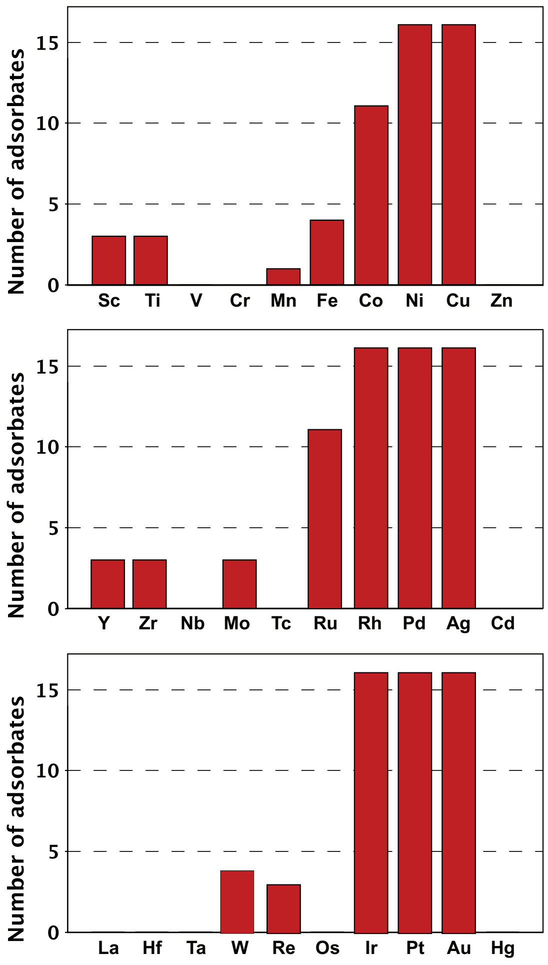


Figure 5. The number of distinct adsorbates for which adsorption energies in the CAMD database exist shown for all the 3d, 4d and 5d transition metals.

formation energies. Those combinations, which have not yet been possible to converge are shown in white. The database was obtained from Professor H L Skriver through its web interface [53].

The database also includes formation energies for over 64 000 alloys containing up to four different elements in simple fcc- or bcc-like crystal structures [44]. These values are calculated using the faster, but somewhat less accurate linear muffin tin orbitals-method (LMTO) [54] except for alloys involving Zn, Cd, Hg, Lu and Si, where the Korringa–Kohn–Rostocker (KKR) method [55] was used to give an accurate description of the low-lying valence or semi-core states. These data can be used to test alloys for stability and phase segregation effects leading to a more accurate estimate of whether or not a given alloy phase can be formed. Figure 8 shows which elements were chosen as components in the combinatorial screening of formation energy for quaternary alloys.

4. Program structure

The VMDF presented here is designed to enable materials researchers or engineers to perform virtual screening of functional materials having a particular combination of physical or chemical properties which should

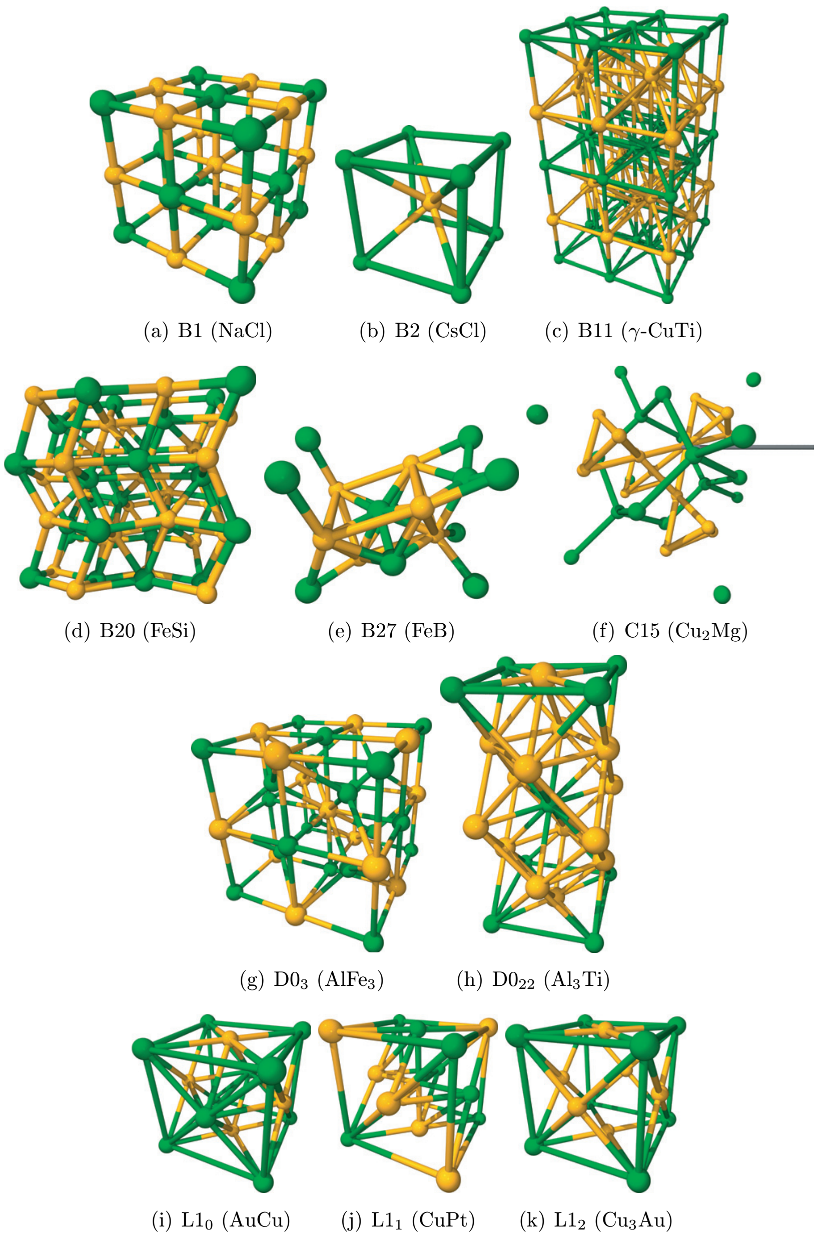


Figure 6. The 11 different crystal structures that the binary alloys have in the presented database of alloy formation energies. The structure names are given in the Strukturbericht nomenclature [69].

be optimized. Furthermore, it enables a user to perform detailed analysis of larger data sets that cannot be analyzed manually or, only with difficulty, analyzed using computer programs that are not specially designed for the problem in question. More extensive analysis of chemical properties and materials design might require programming of additional analysis modules, but these are easy to implement in the presented framework. In this section, the framework that speeds up the development process is presented.

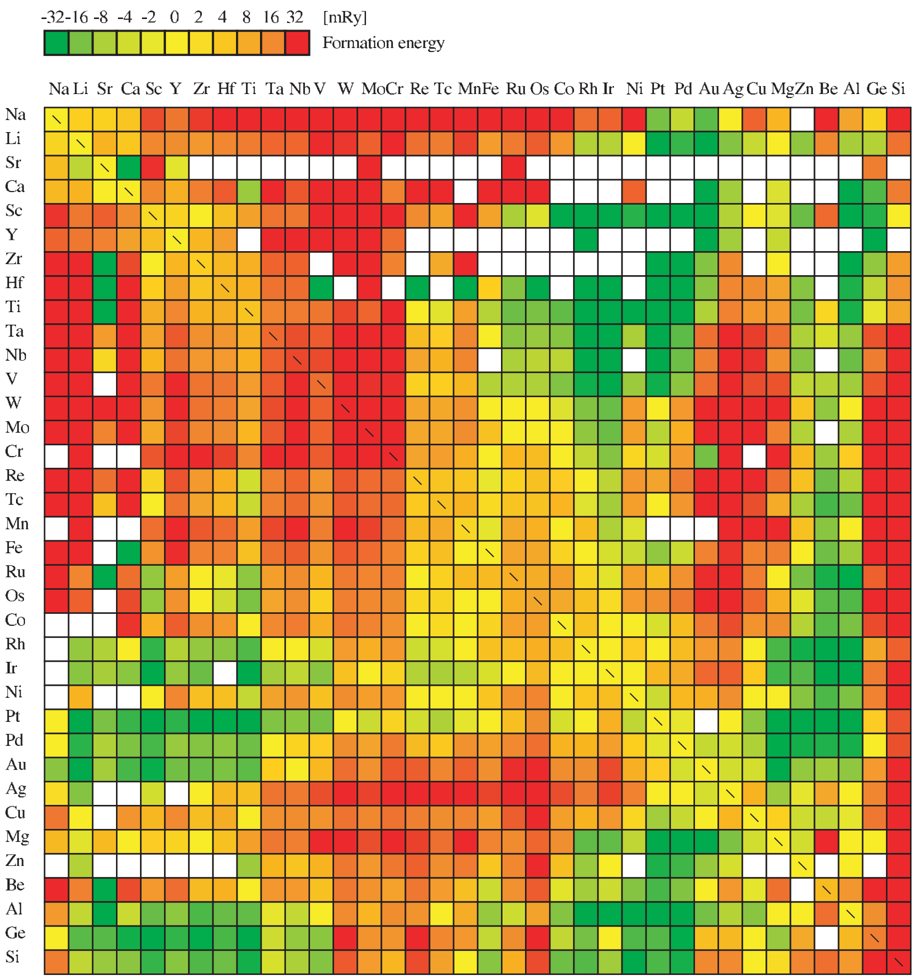


Figure 7. Formation energy for binary alloys having the composition AB_3. All alloys have the Strukturbericht crystal structure $L1_2$. The stability is calculated using EMTO with a GGA xc-functional and is given in mRy atom^{-1}. The chemical elements are ordered accordingly to the Pettifor stringing of the elements [70].

The program for carrying out virtual screening for materials design consists of two parts: the materials design framework (MDF) that abstracts how data sets are stored physically on the disc and provides an application programming interface (API) in which filters, analysis tools and visualization tools can use to perform the screening process. Virtual screening models can be built in a GUI, which provides the means to modify and run the model interactively. The different layers in the program are shown in figure 9 and will be described in more detail in the following discussion.

The MDF enables the user to combine data from multiple data sources, which could be, for example, databases containing theoretical values of chemical properties, databases with experimental data and model-data derived from models based on parameterizations of chemical or thermodynamic properties.

The framework operates with three different types of objects.

Data sources. Objects that feed *data set*s into the model. Data sources come in two different flavors: *data sources* that provide enumerated data, e.g., existing data sets loaded from a database, and *generators*

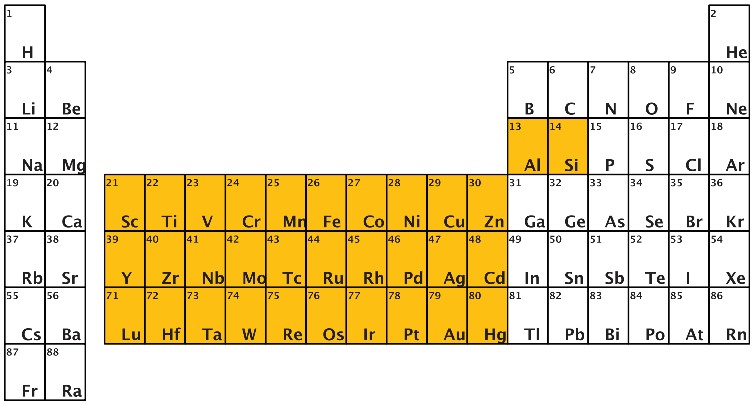


Figure 8. The four components of the quaternary alloys in the database containing LMTO formation energy energies for quaternary alloys [44] are selected from the set of elements colored on this Periodic Table.

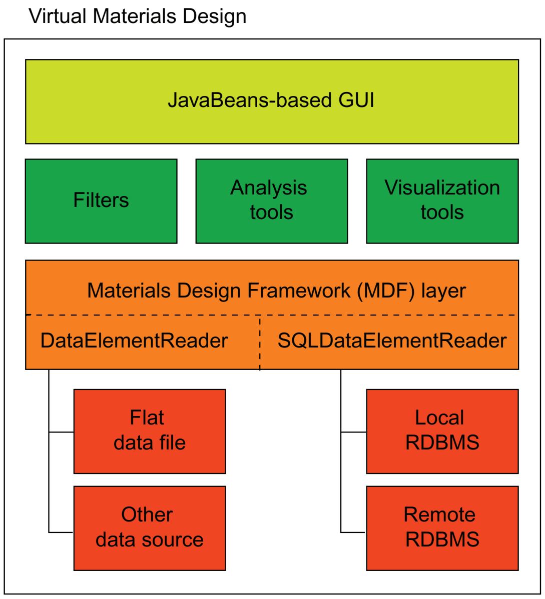


Figure 9. Schematic view of the program structure. The program has a layered structure with the storage layer on the bottom and then more and more data-abstract layers upwards to the top-layer which the user interacts with.

that have to be inquired in order to access information about certain materials because no enumeration is possible. An example of a generator is one which gives the formation energy for oxides of a chemical element at a given temperature calculated from the parameterization of the formation energy as a function of temperature.

Filters. Objects that take the output of a data source (or of another filter if they are linked) and evaluate each element in the data set against predefined criteria. Only data elements that meet the criteria are allowed to pass and there is the further option to add more data fields about the material in question.

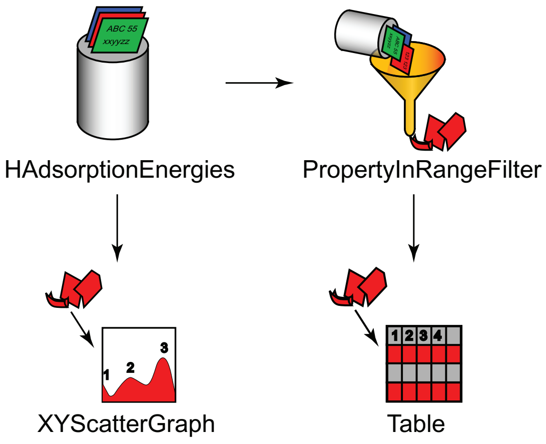


Figure 10. Simple example of a virtual screening. The data set 'HAdsorptionEnergies' containing theoretical values for H adsorption energies on binary alloys is loaded and passed to a visualizing tool that plots the data set in a scatter graph and to a filter named `PropertyInRangeFilter` that only accepts data elements which have a certain property in the selected range. The data elements that pass the filter are then presented in a table.

In general, data elements of a data set are fed to filters in a non-deterministic order so filters have no information available to them about the current position in the data set. In special cases a filter can store a local copy of the data set in order to perform certain tasks, for example, sorting a data set.

Visualization tools. Objects of this type can be connected to the output of a chain of filters or directly to a data source to visualize the data set. A multitude of different ways to visualize a data set can be imagined, such as the data set presented in a table, shown as a scatter-graph or as a histogram. The illustrations in the first part of this paper are examples of ways to visualize data sets.

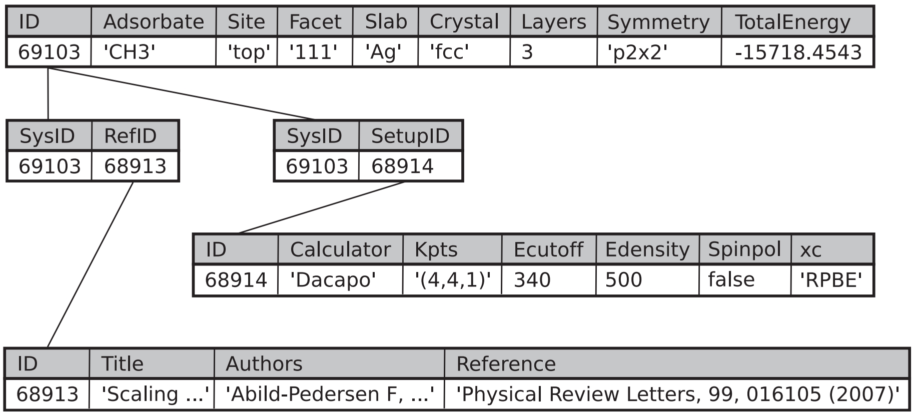
As previously mentioned, data elements in a data set are in non-deterministic order, this is because ordering a data set according to the value of a property does not make sense except when presenting the data set in a table for the user. Due to the fact that filters can add or remove fields in individual data elements, care is taken to make sure that filters cannot change instances of the data sets seen by objects upstream in the model. This makes it easy to run the evaluation of the model in parallel since a data element, as soon as it has passed one object, can be fed into the next object in the chain. The downside is the extra cost of memory. It also makes it easy to implement one-to-many relationships: the output from one data source or filter is fed to one or many filters. Figure 10 illustrates this.

The virtual screening program is implemented in Java running on Sun's JavaSE 5 SDK [56]. The decision to use Java was made to make the program portable. Java offers a versatile program library containing packages to build advanced user-interfaces, and a multitude of packages offers functionalities such as embedded database systems; and molecule visualization is also available from other sources.

The GUI takes care of the user interaction, the editing of models and the display of plots. For the editing of object properties, Java's built-in functionality for reflection is used to discover the properties of data sources, filters and visualization tools. Reflection enables a program to discover properties about classes *at* run-time, thereby not requiring any hardcoded information about classes.

Reflection works best when the JavaBeans design pattern [57] is applied in the class-design, therefore the classes in the MDF are designed following the JavaBean design-pattern.

The only configuration the program needs is an XML file containing the names of the classes of the data sources, filters and visualization tools. Everything after this point can be discovered at run-time using reflection.



ID	Adsorbate	Site	Facet	Slab	Crystal	Layers	Symmetry	TotalEnergy
69103	'CH3'	'top'	'111'	'Ag'	'fcc'	3	'p2x2'	-15718.4543

SysID	RefID
69103	68913

SysID	SetupID
69103	68914

ID	Calculator	Kpts	Ecutoff	Edensity	Spinpol	xc
68914	'Dacapo'	'(4,4,1)'	340	500	false	'RPBE'

ID	Title	Authors	Reference
68913	'Scaling ...'	'Abild-Pedersen F, ...'	'Physical Review Letters, 99, 016105 (2007)'

Figure 11. Schematic illustration of how a subset of the data is organized in tables in the relational database.

Most data sets are stored in the embedded SQL database server, a relational database management system (RDBMS). For the embedded RDBMS the Apache Derby database version 10.3 is used [58]. It offers a fast Java RDBMS that implements a large part of the SQL standard. The RDBMS can be embedded into an application and the data files can be packed in a JAR-archive and be distributed along the main application. Access to the database is provided through JDBC which is Java's mechanism for accessing relational database systems.

Creation of connections to the embedded RDBMS is handled by a separate class. This pattern allows one to switch to the back-end RDBMS easily, as long as data sources and other classes only use standard SQL. Unfortunately, database systems do not always implement the whole SQL standard and therefore database-specific dependencies can still be found in the modules.

As illustrated in figure 9 data sets can be stored in many different ways, e.g., in the embedded RDBMS, in a local text file or in an off-site database server. The data set is loaded using a derivative of either `SQLDataElementReader` or `DataElementReader`, depending upon whether access to a SQL database server is needed or not. After loading, access to data sets happens in a uniform way defined by the MDF, independent of the data origin.

Due to the fact that information stored in different data sets is often very heterogeneous, it is necessary to store each data set in an individual table in the embedded RDBMS. An example of which columns a data set contains on-disc is shown in figure 11. Relationships between tables are used to find additional information about a particular property, e.g., relationships implemented using foreign keys are used to find information about computational setup and references to where the data originates for the row containing the actual material property. This is important because one data set can contain data coming from multiple sources, therefore it is vital that the origin of a data element can be traced back to where it was published and how it was obtained.

Each data element is wrapped in an instance of the `DataElement` class. This class inherits from the `MaterialTemplate` class which offers basic functionality to store values associated with keys and to be able to retrieve the value associated with the given key. Instances of the `MaterialTemplate` class are also used when searching for data elements matching an incomplete description of a material, this could be all alloys having a certain crystal structure. `DataElement` imposes some constraints on the information stored, the data element must, for example, contain a reference to where the data comes from, a name to be presented to the user and a list of the elements that the material contains.

The full class diagram can be seen in figure 12. Data sources are implemented as instances of either `MaterialList` or as derived classes of `MaterialGenerator`. Sources where the data can be enumerated are instances of `MaterialList` and we can therefore iterate through all elements, data sources that can only be

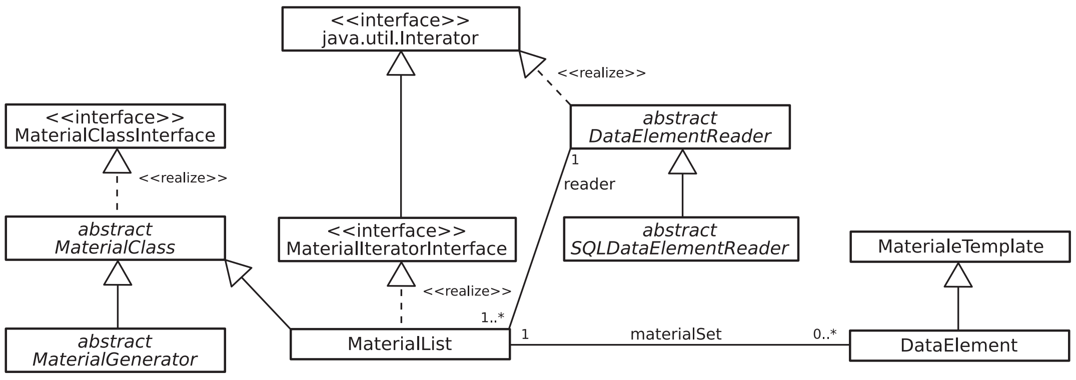


Figure 12. UML class diagram of the MDF-package.

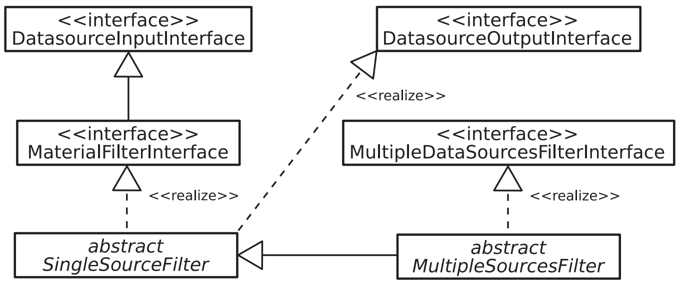


Figure 13. UML class diagram of the `filters`-subpackage of the MDF.

queried for data about a certain material or class of materials inherit from `MaterialGenerator`. An instance of `MaterialList` can either be constructed using an instance of a class inheriting from `DataElementReader` or be initialized to contain the output from a filter. Classes derived from `DataElementReader` are used to read data sets from files or over the internet, where the more specialized class `SQLDataElementReader` is used when loading data sets from a RDBMS.

Data sources, filters and visualization tools built on the MDF must also implement either the `DatasourceInputInterface` and/or the `DatasourceOutputInterface`. By inheriting from these interfaces the modelbuilder knows which modules can be connected. The class diagram for data sources, filters and visualization tools is shown in figure 13.

A filter or analysis tool which is iterating over the data set one element at a time, can inherit from the class `SingleSourceFilter` and get a lot of functionality by inheritance. In this way it becomes possible to implement a new filter by just overriding one method concentrating on the new functionality, thereby using the methods inherited from `SingleSourceFilter` to handle all the tasks common to all filters. Filters combining input from multiple data sources can inherit from the class `MultipleSourcesFilter`. Multiple data sources are handled though this filter by iterating over elements in the primary data source and using these elements in queries to the secondary data sources. In this way, a data element from one `MaterialList` can get additional information added from other iterable `MaterialLists` or from a `MaterialGenerator`.

Figure 14 shows a more realistic virtual screening model, where three filters are applied to the data set coming from the data source. The resulting data set is then plotted as a checkerboard plot and in a table.

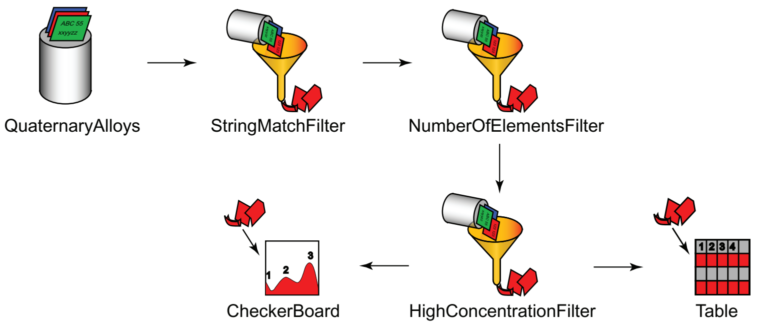


Figure 14. Example of a virtual screening. The data set comes from a data source named 'QuaternaryAlloys' contain the data presented in [44]. After being loaded the data elements are filtered, so only those having a fcc-like crystal structure pass, then only binary alloys are allowed to pass, the last filter is one that selects binary alloys containing elements with the ratio 0.25 and 0.75, respectively. Finally, the data are plotted in a checkerboard graph and shown in a table.

5. Example 1. Screening for mercury absorbing materials

In this example, it will be shown how the proposed framework for performing virtual screening in a materials design context can be applied. The methodology is applied to the problem of searching for a material that potentially can absorb gaseous Hg from a gas stream.

Hg is naturally present in coal and crude oil in minute amounts. Given the large quantities of hydrocarbons that pass through the chemical plants treating these chemicals, even traces of Hg will add up. It is estimated that the largest aggregate source of Hg pollution in the US are coal-fired power plants, which in 1999 were responsible for 40% of the total Hg emission [59].

Hg and other heavy metals are toxic to humans and animals and cause severe pollution of the environment. Furthermore, Hg will poison the active sites of a number of catalysts used in reactions, where trace amounts of Hg can be found in the feed-gas stream. These processes include, for example, the methanation process and the production of synthetic natural gas. The ability to remove Hg from the feed-gas stream of a chemical plant or from exhaust gas from combustion of coal or crude oil is therefore crucial for the environment and to the operators of chemical plants alike.

A solution for the removal of gaseous Hg is proposed by which a regenerable absorber material will absorb the Hg from a gas stream when it is passed over the material, forming an alloy in the process. In the following section 5.1, some key properties that such a material should have will be described. We shall now turn our attention to describing, in more detail, how these criteria are defined.

5.1. Screening for potential candidates

Candidate materials that can absorb Hg and, at the same time, are stable are required to fulfill a long list of constraints. Several of these constraints are not known exactly, and must therefore be established from experience. The constraints that we chose for the candidate alloys are listed below.

(i) Candidate has a negative formation energy.
(ii) Candidate contains Hg.
(iii) Candidate is stable to oxidation at the given reaction conditions.
(iv) Candidate must have a more negative formation energy than the critical formation energy at the partial pressure of Hg at the entrance of the absorber.
(v) Candidate must have a more negative formation energy than the critical formation energy at the partial pressure of Hg at the exit of the absorber (1/100 of the Hg pressure at the entrance).
(vi) If candidates in the result set are polymorphic only the most stable phase passes.

Table 1. Fractional gas composition of the gas stream coming from a gasifier at a temperature of $T = 473\,\mathrm{K}$ and a total pressure of $p = 3\,\mathrm{MPa}$.

	X_i
H_2	0.27
CO	0.62
CO_2	0.02
H_2O	0.04
H_2S	<1 ppm

The choice of absorbing 99% of the Hg in (v) is somewhat arbitrary, but appears to be an reasonable design goal. In the following, we describe in more detail how these criteria are defined and implemented.

5.2. Reaction conditions

The presented thermodynamic calculations are for a system with a total pressure, p_{total}, of 3 MPa, a temperature, T, of 473 K and a gas composition as shown in table 1.

5.3. Critical formation energy

In order for the alloy to be able to absorb gaseous Hg from a gas stream, the alloy after absorption (which contains Hg) must be more stable than the alloy without. In order for the alloy to be able to absorb Hg the stability of the alloy containing Hg must be higher by an amount here called the *critical formation energy*, $E_{\mathrm{F}}^{\mathrm{critical}}$, with respect to the alloy without. The critical formation energy can be found by asserting that the absorption of Hg(g) happens in the following reaction:

$$x\mathrm{Hg(g)} + yM(\mathrm{s}) \rightleftarrows \mathrm{Hg}_x M_y(\mathrm{s})\,. \tag{1}$$

The change in enthalpy during reaction, $\Delta H_{(1)}$, is

$$\begin{aligned}\Delta H_{(1)} \approx \Delta E &= \underbrace{E_{\mathrm{Hg}_x M_y(\mathrm{s})} - (xE_{\mathrm{Hg(s)}} + yE_{M(\mathrm{s})})}_{E_{\mathrm{F}}(\mathrm{Hg}_x M_y(\mathrm{s}))} + \underbrace{(xE_{\mathrm{Hg(s)}} - xE_{\mathrm{Hg(g)}})}_{-x\,E_{\mathrm{cohesive}}(\mathrm{Hg})} \\ &= E_F(\mathrm{Hg}_x M_y(\mathrm{s})) - x\,E_{\mathrm{cohesive}}(\mathrm{Hg}),\end{aligned}$$

and the change in entropy is:

$$\Delta S_{(1)} = S_{\mathrm{Hg}_x M_y(\mathrm{s})} - (xS_{\mathrm{Hg(g)}} + yS_{M(\mathrm{s})}),$$

where $S_{\mathrm{Hg}_x M_y(\mathrm{s})} = S_{\mathrm{vib}} + S_{\mathrm{mixing}}$. S_{vib} is assumed to cancel the vibrational contribution to $S_{M(\mathrm{s})}$. The error introduced by neglecting vibrational entropy for the metal and the alloy is difficult to estimate, but it can be noted that the entropic contributions to the Gibbs free energy at room temperature in, e.g., Cu, Ag and Au are 0.05, 0.07 and 0.08 eV atom^{-1}, respectively, and that the property we aim for is obtained as the difference between two such terms. Although the vibrational contribution to the thermodynamic properties of different alloys can, in principle, vary quite a bit, the error thus appears relatively bounded. The mixing entropy, S_{mixing}, we (conservatively) assume to be 0.0 eV. Presumably there will be some mixing entropy contribution, at least at low concentrations of Hg in the absorber. In this case, we are underestimating the actual ability of the potential absorber materials. However, if we focus on the relevant regime of alloys with a relatively high concentration of Hg corresponding to the point when the material has to be recycled, the differential mixing entropy contribution for an alloy with 25% Hg is maximally of the order of $kT\ \ln 3$, which is 0.043 eV atom^{-1} under reaction conditions. We thus estimate the omitted entropy terms to be an order of magnitude smaller than the variations in formation energies between the different alloys, an order of magnitude smaller than the possible variations of the oxygen chemical potential, smaller than the error introduced by only looking at structures with small unit cells, and smaller than the errors introduced by the limited accuracy of the employed exchange-correlation functional. We therefore do not expect the omitted entropy terms to significantly change our conclusions. The change in entropy for reaction (1) thereby reduces to

$$\Delta S_{(1)} \approx -x S_{\mathrm{Hg(g)}},$$

where $S_{\mathrm{Hg(g)}}$ is the entropy of Hg in the gas phase at standard conditions corrected for the changed temperature and pressure. The Gibbs free energy is then given by

$$\Delta G \approx E_{\mathrm{F}}(\mathrm{Hg}_x M_y(s)) - x\, E_{\mathrm{cohesive}}(\mathrm{Hg}) + x\, T\left(S^{\circ}_{\mathrm{Hg(g)}} - k \ln \frac{p_{\mathrm{Hg}}}{p^{\circ}}\right).$$

For alloys that can be formed $E_{\mathrm{F}}(\mathrm{Hg}_x M_y(\mathrm{s}))$ is negative. The cohesive energy of Hg, E_{cohesive}, is found to be 0.67 eV atom^{-1} [60].

The critical formation energy is the alloy formation energy (per Hg atom and from pure solid metals at 0 K) at which Hg is equally stable in the gas phase and absorbed in the alloy at the given temperature. This means that $\Delta G = 0\,\mathrm{eV}$. Assuming the specific heat capacities of the alloys cancel the specific heat capacities of the pure metals, the critical formation energy can thus be determined as

$$E_{\mathrm{F}}^{\mathrm{critical}}(\mathrm{Hg}_x M_y) = x E_{\mathrm{cohesive}}(\mathrm{Hg}) - xT\left(S^{\circ}_{\mathrm{Hg(g)}} - k \ln \frac{p_{\mathrm{Hg}}}{p^{\circ}}\right). \tag{2}$$

At 473 K the entropy contribution, $-TS$, from Hg at a partial pressure at the inlet, p_{Hg}, of 0.03 Pa is -1.52 eV. Assuming that the partial pressure of Hg(g) after passing the absorbing alloy is 1/100 of the partial pressure at the inlet, then the entropy contribution from Hg is -1.69 eV.

In a unit cell containing four atoms of which one atom is a Hg atom ($x = 1$, $y = 3$ in reaction (1)) the critical formation energy of the alloy at the start of the absorber is

$$E_{\mathrm{F}}^{\mathrm{critical}}(\mathrm{inlet}) = -0.85\,\mathrm{eV} N_{\mathrm{Hg}}^{-1}.$$

After 99% of the Hg has been absorbed the critical formation energy is

$$E_{\mathrm{F}}^{\mathrm{critical}}(\mathrm{outlet}) = -1.04\,\mathrm{eV} N_{\mathrm{Hg}}^{-1},$$

where N_{Hg} is the number of Hg atoms in the alloy unit cell. Figure 15 shows a contour plot of ΔG at a partial pressure of Hg of 0.0003 Pa as a function of temperature and $E_{\mathrm{F}}^{\mathrm{critical}}$. The contour line for $\Delta G = 0.0\,\mathrm{eV}$ can be used to find the critical formation energy of Hg-containing alloys at different temperatures at the partial pressure at the end of the absorber. For increasing temperatures the critical formation energy also increases meaning that the Hg-containing alloy must be increasingly stable in order to absorb Hg as the temperature is increased.

5.4. Stability against oxidation

At the gas composition described in table 1 the following reactions could occur:

$$\mathrm{H_2O} + \mathrm{CO} \rightleftharpoons \mathrm{CO_2} + \mathrm{H_2} \qquad \text{(WGS)} \tag{3}$$

$$\mathrm{H_2O} + M \rightleftharpoons M\mathrm{O} + \mathrm{H_2} \tag{4}$$

$$\mathrm{CO_2} + M \rightleftharpoons M\mathrm{O} + \mathrm{CO} \tag{5}$$

$$\mathrm{H_2O} \rightleftharpoons \tfrac{1}{2}\mathrm{O_2} + \mathrm{H_2} \tag{6}$$

$$\mathrm{CO_2} \rightleftharpoons \tfrac{1}{2}\mathrm{O_2} + \mathrm{CO} \tag{7}$$

$$\tfrac{1}{2}\mathrm{O_2} + M \rightleftharpoons M\mathrm{O} \tag{8}$$

M denotes either a pure metal or an alloy, 'WGS' indicates that this reaction is the *water gas shift* reaction. Reactions (6) and (7) are considered because they define a range for the chemical potential of oxygen that can be used directly with reaction (8). This simplifies the description of the most likely oxidation process, which is likely to be (4) and (5).

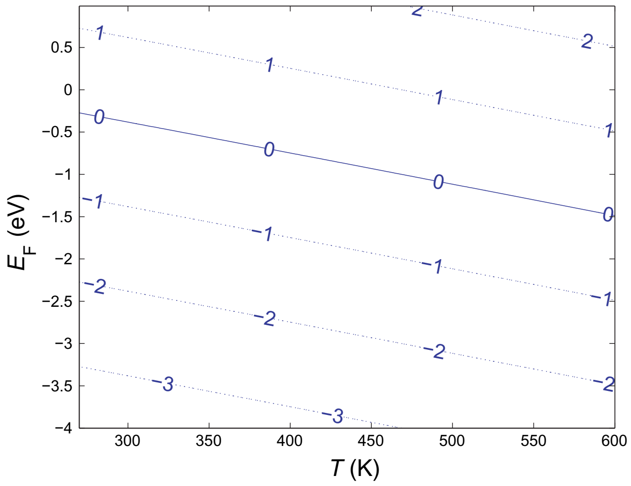


Figure 15. Contour plot of ΔG as a function of temperature, T, and formation energy of the alloy, E_F for alloys contains 1 Hg atom per unit cell. The alloys are thus of the form M_3Hg. The contour at $\Delta G = 0.0 \frac{\text{kJ}}{\text{mol}}$ shows the critical formation energy at a partial pressure of Hg of $p_{Hg}(\text{out}) = 0.0003$ Pa for alloys as a function of temperature.

5.4.1. Chemical potential of oxygen in the gas phase. To analyze whether the absorbing material will oxidize when the gas stream is led over it, it is useful to find the chemical potential of free oxygen in the system. We do this by calculating the O_2 pressure at equilibrium of reactions (6) and (7), respectively, using

$$p_{O_2}^{1/2}(6) = \frac{p_{H_2O}}{p_{H_2}} \exp\left[-\Delta G^\circ_{(6)}/(k_B T)\right],$$

$$p_{O_2}^{1/2}(7) = \frac{p_{CO_2}}{p_{CO}} \exp\left[-\Delta G^\circ_{(7)}/(k_B T)\right],$$

where ΔG°_6 and ΔG°_7 are the standard Gibbs free energies of reactions (6) and (7) and k_B is Boltzmann's constant. Subsequently, the chemical potential is determined as

$$\mu_{O_2} = k_B T \,\ln\left(\frac{p_{O_2}}{p^\circ}\right),$$

Two different partial pressures are obtained due to the inequlibrium of the gas composition. At the partial pressures from reactions (6) and (7), the chemical potential (μ_{O_2}) of oxygen in the gas stream at a temperature of 473 K is in range of −5.28 to −4.75 eV.

5.4.2. Formation energies of oxides. To test for stability of an alloy with respect to oxidation the formation energies of the corresponding oxides are needed. We separate the oxidation of alloys into two steps: decomposition of the alloy into pure elements and then the oxidation of pure elements, in order to obtain the formation energies of oxides from the data in the database. We first look at the formation energies of the oxides of the pure elements, then the decomposition energies of alloys (minus their formation energies) are considered to connect pure elements with alloys. For the formation energies of the oxides of the pure elements, we consider the reaction

$$yM + xO_2 \rightleftharpoons M_yO_{2x}$$

Table 2. Summary of the number of candidate alloys that pass each additional screening criterion as described in section 5.1.

Criterion	Candidates
Database	64 117
Stable alloys	31 041
Contains Hg	1319
Stable to oxidation	122 or 116
Formation energy less than −0.85 eV	32
Formation energy less than −1.04 eV	24
Polymorfic alloys discarded	14

then $\Delta_F G_{\mathrm{oxide}}$ is the formation free energy of the oxide of the pure chemical element M. Under the given reaction condition the free energy of formation is given by

$$\Delta_F G^{\mu_{O_2}}_{\mathrm{oxide}} = \Delta_F G^{\circ}_{\mathrm{oxide}} - x\,\mu_{O_2}.$$

The free energies of formation for pure metal oxides as a function of temperature have been parameterized in [61].

For all metals their oxides becomes less stable when the temperature increases. It is desirable for the alloys we are searching for that they are more stable in the alloy-form than as an oxide. Therefore, one might expect that the reaction temperature should be raised. However, as shown in section 5.3, an increased temperature gives an increased critical formation energy, and one therefore has to balance the two effects when choosing the reaction temperature.

Considering the second step of oxidation of an alloy, after a quaternary alloy decompose into four pure elements, there are 15 possible oxidation events. The reaction energies of these events could be calculated by linear combination of the formation energies of oxides of the four elements. We then select the reaction energy $\Delta G_{\mathrm{oxidation}}$ of the most energetically favorable event in the subsequent calculation. The formation energy of oxides from an alloy is then calculated by summing up the decomposition energies and $\Delta G_{\mathrm{oxidation}}$.

5.5. Candidates

A screening with the criteria described in section 5.1 was performed on the database containing the formation energies for 64 117 binary, tertiary and quaternary alloys [26, 44] described in section 3. The database contains alloys having five different fcc- and bcc-like crystal structures. Table 2 summarizes the number of alloys passing each criterion described in section 5.1. The two numbers given for the number of alloys that are stable against oxidation are for the two extrema of the interval of the chemical potential of oxygen. It is observed that even the relatively large change in the oxygen chemical potential does not drastically change the number of available alloys. In fact, after the critical formation energy is taken into account, the number of available candidates is independent of which of the two chemical potentials were used in the oxide stability part of the screening. In table 3, the 14 candidates that are able to absorb Hg and are stable to oxidation are presented.

5.6. Discussion

The steps necessary to carry out a virtual screening for candidate materials that can absorb gaseous Hg from a gas stream have been described. By applying the described criteria it was possible to reduce the number of Hg-containing candidates from 5581 down to 14.

When the list of criteria described in section 5.1 are applied only candidates pass that:

- Do not oxidize under the given reaction conditions.
- Have a formation energy that is more negative than the critical formation energy. This insures that the alloy containing Hg is stable both at the partial pressure of Hg found at the beginning of the absorber and at 1/100 of this pressure, which we assert is the partial pressure at the exit of the absorber.

Table 3. The alloy candidates that pass all the criteria described in the text.

Candidate	Struct.	Formation energy (eV)
$HfPt_2Hg$	BCC	−3.27
HfPdPtHg	BCC	−3.05
HfPdRhHg	BCC	−2.59
HfIrPdHg	BCC	−2.33
HfIrRhHg	BCC	−2.29
$HfIr_2Hg$	BCC	−1.89
Hf_2IrHg	BCC	−1.80
CdHfPtHg	BCC	−1.70
Hf_2PtHg	FCC	−1.54
HfPtRhHg	FCC	−1.50
$HfNi_2Hg$	BCC	−1.42
Pd_2ZnHg	FCC	−1.14
HfPtRuHg	FCC	−1.12
CdPdPtHg	BCC	−1.05

Table 3 summarizes the result of applying the described 'filters' to a database of quaternary alloys. Only 14 candidates satisfies the proposed criteria for stability and ability to absorb Hg. They are possible candidates for absorbtion of Hg from a gas stream and could be tested experimentally.

The scanned alloy database contains quaternary alloys in only five different bcc- and fcc-like crystal structures (see [26]). Because of this it can be expected that some of the alloys exist in structures that are more stable than those tested. The most stable candidates are given in table 3, they are the most likely to pass the described criteria.

The reaction conditions described in table 1 are used for the screening. If the temperature is lowered the alloys ability to absorb Hg increases but an opposite effect also plays a role, as the temperature is lowered the oxides becomes more stable, so the absorber will be more likely to oxidize.

The above analysis is meant primarily as a demonstration of the capabilities of the VMDF. Indeed a number of improvements are important to consider in order to find new realistic transition metal alloy Hg absorbers. It would, for example, be very interesting to look at whether the Hg adsorbs more strongly on surfaces [62] than in the bulk which we have implicitly assumed here. It would likewise be interesting to address in more detail the kinetics of the Hg absorption. In the present study we analyze only the thermodynamics, and it is of course a prerequisite that the thermodynamics are favorable in order for the absorber to work. An additional prerequisite is whether the kinetics at the given reaction temperature are favorable enough for the reaction to occur. The limited number of alloys and the very simple crystal structures in the database give some constraints to the applicability of the results, and it would very much be worth while to include more metals, structures and complexity.

Finally, it could be argued that in order to keep the Hg in the absorber, the Hg-containing alloy should actually have a segregation energy (into metallic Hg and the most stable linear combination of pure metals and binary and ternary alloys of the other constituents) of the order of the critical formation energy. This segregation energy is shown in table 4 for the three candidate Hg-alloys, which are stable against segregation. If this criterion was to be enforced, we could conclude that no relevant combination of transition metals in our database would be able to absorb Hg under the given conditions.

6. Example 2. The methanation reaction

To illustrate another use of the VMDF we have applied it to the methanation reaction:

$$CO + 3H_2 \leftrightarrows CH_4 + H_2O$$

Table 4. Candidates sorted according to the stability of the alloy relative to the most stable linear combination of alloys and pure elements excluding Hg (ΔE_F). The three alloys are those alloys from table 3 that are most stable with respect to segregation.

Candidate	Struct.	Formation energy (eV)	ΔE_F (eV)
HfPdPtHg	BCC	−3.05	−0.10
CdPdPtHg	BCC	−1.05	−0.07
HfPdRhHg	BCC	−2.59	−0.02

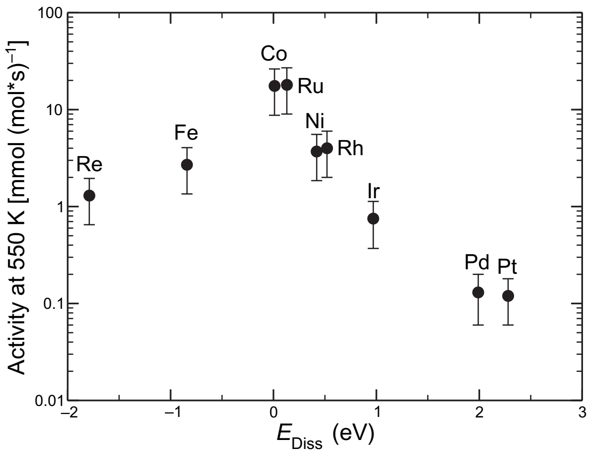


Figure 16. The activity of transition metals for the CO methanation reaction plotted as a function of the dissociative adsorption energy of CO relative to adsorbed CO on the metal. Adapted from [31].

This is one of the classic reactions in heterogeneous catalysis [63], and it is used, for example, to remove CO from the hydrogen feed gas in ammonia plants. The reaction has been studied intensely both experimentally [64, 65] and theoretically [17, 66, 67].

The rate limiting elementary reaction step in methanation is the dissociation of the strongly bound CO molecule. The dissociation energy of CO on a stepped surface, E_{diss}, has proven to be an excellent descriptor for the reactivity of a methanation catalyst [17, 31]. Figure 16 shows the experimentally measured CO methanation activity of a number of transition metals plotted as a function of the calculated CO dissociation energy.

Screening for methanation catalysts has previously been carried out [17, 31] and promising candidates were identified and subsequently tested experimentally [68]. In this section, we describe how the VMDF can be applied to the problem of finding new methanation catalysts. The virtual screening presented here correlates data from two sources in order to find the most promising candidates.

To generate the set of methanation-catalyst candidates to be investigated, the descriptor is generated for binary alloys of the form

$$A_{1-c}B_c, \quad \text{where } c \in \left\{0, \tfrac{1}{4}, \tfrac{1}{3}, \tfrac{2}{4}\right\}. \tag{9}$$

The descriptor is generated as the weighted average of the value of E_{diss} for the pure elements going into the alloys. The descriptor used is the dissociative adsorption energy of CO relative to the adsorption energy of molecular CO as used in [31]. The descriptor for the alloys is thus calculated as [31]

$$E_{\text{diss}}^{A_{1-c}B_c} = (1-c)\,E_{\text{diss}}^{A} + c\,E_{\text{diss}}^{B}.$$

Table 5. Table summarizing the number of candidate materials that pass each screening criterion for the methanation reaction.

Criterion	Candidates
Initial set	153
Activity	39
Stability	10

The descriptor is generated for all alloys that can be formed of the form in (9) where A and B are taken from

$$A, B \in \{\mathrm{Ni, Pd, Pt, Co, Rh, Ir, Fe, Ru, Re}\}$$

To screen the set of candidates for good leads, both activity and stability criteria are now defined. The following two criteria are implemented in this particular screening:

- the candidate alloys must be reactive for the methanation reaction;
- the candidate alloys must be stable in the specific composition.

We shall now examine the two criteria in more detail. The maximal activity for methanation catalysts is found to be around $E_{\mathrm{diss}} = 0.06\,\mathrm{eV}$, therefore the top of the activity 'volcano' is in the vicinity of where Co and Ru are found [31].

As descriptor for catalyst activity the absolute distance to the optimal point on the experimental volcano is chosen. Because the accuracy of the interpolated values for the descriptor is lower than the accuracy of the full DFT calculations, the screening for quality has to be reasonably lax in order not to throw away candidates that could be active. In this study, the acceptable quality criterion is defined to be:

$$\left| E_{\mathrm{diss}}^{\mathrm{candidate}} - E_{\mathrm{diss}}^{\mathrm{optimal}} \right| \leqslant 0.30\,\mathrm{eV}\,.$$

This corresponds to approximately half the distance between neighboring elements in the Periodic Table. Only candidates with an activity descriptor within this interval pass the screening. Several different tests can be thought of for testing the stability of a catalytic material. Particularly important for alloy catalysts is the question of whether or not it will segregate into its pure metal constituents.

In this study of candidates for the methanation reaction only the alloy formation energy is considered. This is a basic quantity describing whether the alloy can be formed or not. Other stability criteria such as stability against formation of oxides and carbides could easily be implemented, as demonstrated for the Hg absorber case.

To test for stability at the current composition, each alloy is tested against the database containing theoretical values for the formation energy of binary alloys calculated using EMTO. For each candidate the database is queried for the most stable crystal structure found for the given candidate's composition. The formation energy for the most stable encountered phase is associated with the given candidate and is used later.

The database contains formation energies of binary alloys in 11 different crystal structures. It is not guaranteed that the physically most stable phase is found in the database, but correlating candidates with the database gives an indication of whether the alloy can be found in the tested phase.

Using the above-mentioned criteria in a virtual screening candidates for the methanation reaction are found. Table 5 summarizes the number of candidates passing each of the proposed quality and stability criteria.

Candidates that pass the activity criterion are only useful if the phase can be synthesized. Table 6 summarizes the candidates that pass both the activity and the stability criteria. In figure 17, the candidates are plotted as a function of the activity and stability descriptors. The interesting candidates are found in the lower left corner of the graph.

Table 6. The 10 candidates that pass both the activity and the stability criterion. Candidates which are pareto-optimal are written in bold.

Candidate	Most stable phase	E_{diss} (eV)	Quality (eV)	Formation energy (eV atom^{-1})
Fe_2Ir_2	FeIr (B2)	0.06	0.005	−0.07
$FeNi_3$	$FeNi_3$ ($L1_2$)	0.11	0.04	−0.09
Co	Co (hcp)	0.01	0.05	0.00
Ru	Ru (hcp)	0.13	0.07	0.00
Fe_3Pt	$PtFe_3$ (DO_3)	−0.06	0.12	−0.12
$FeRh_3$	$FeRh_3$ ($L1_2$)	0.18	0.12	−0.08
$ReRh_3$	$ReRh_3$ (DO_{22})	−0.06	0.12	−0.04
Fe_2Rh_2	FeRh (B2)	−0.16	0.22	−0.12
$ReIr_3$	$ReIr_3$ (DO_{22})	0.28	0.22	−0.08
Fe_2Ni_2	FeNi ($L1_0$)	−0.21	0.27	−0.06

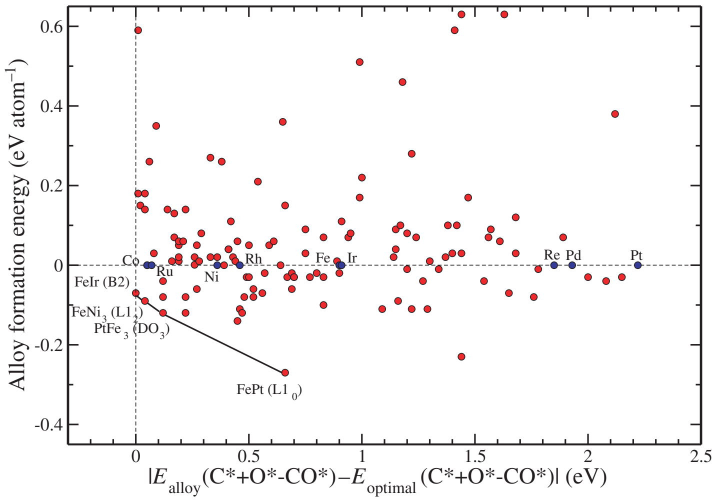


Figure 17. Methanation catalyst candidates plotted as functions of the activity descriptor, the distance in CO dissociation energy to the top of the experimental volcano-curve, and the stability descriptor, the alloy formation energy of the most stable alloy of the right composition in the database. The Pareto-optimal set is shown and the alloys are labeled with their composition and most stable structure.

Andersson *et al* [31] conducted a screening study of the methanation reaction, there candidates were plotted as a function of activity and cost of the alloy components. Comparing the candidates from table 6 with the Pareto-optimal candidates proposed in that study it is seen that both IrFe and $FeNi_3$ are found to have favorable qualities in both studies. The Pareto-optimal candidates CoRu and Co_3Ru, also found when using cost as a descriptor, are not found to be stable in the present study (see figure 18). An interesting new finding here is thus that the proposed $FeNi_3$ catalyst is a relatively very stable alloy. This could seem to indicate, that high alloy stability is one of the reasons why the computationally discovered $FeNi_3$ methanation catalyst could be turned into a technical catalyst [68].

7. Summary. conclusion

We have here presented an application framework for performing virtual screening in materials science. The framework was designed with flexibility and extensibility in mind. At present only a limited number of filters

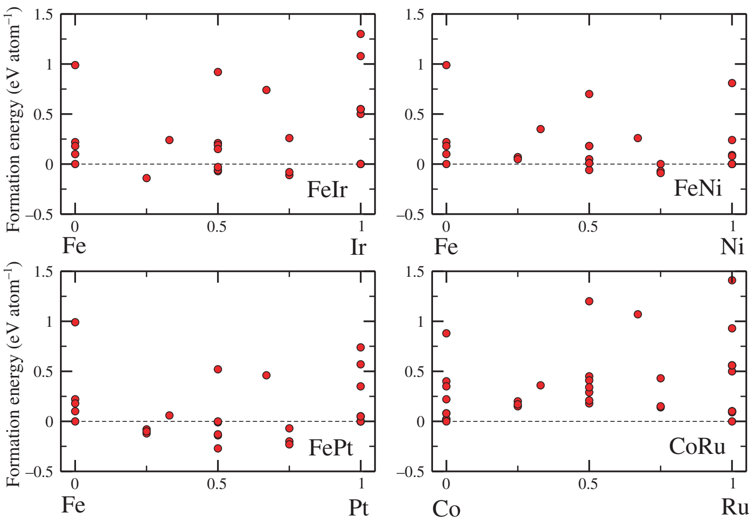


Figure 18. Stability of binary alloys with different compositions in a number of different crystal structures.

and analysis tools have been implemented in the described framework. However, as the framework was created with extensibility in mind, it should be a simple task to create new filters and analysis tools as the need arises. We believe that because of this, we will see it applied to a number of other applications in the future, beyond those presented here.

The process of collecting calculated materials properties has just begun. This issue constitutes an important challenge, particularly because of problematic issues relating to the consistency and compatibility between different datasets. We have presented here a database of materials properties containing more than 81 500 data points. If the process of describing a calculation could be automated, large and versatile databases could be created more easily than at present. The automatization could be done by making the documentation of the calculation an integrated part of the used simulation environment. This is, however, by no means a simple task, as care must be taken that all relevant types of calculations are describable. Effort might also need to be done to introduce some kind of a quality measure, such that calculations are marked as trustworthy only after the data has achieved a reasonable and well-defined level of quality.

Combining theoretical data obtained from, for example DFT calculations with knowledge of trends and qualitative models is now possible and will be used for studies much more frequently in the future.

Acknowledgments

We thank Professor Jens K Nørskov for encouraging the construction of the VMDF and for his constant support and inspiration. We thank Professor Hans L Skriver for many helpful discussions, and for providing us with his database of 8611 formation energies for binary alloys that were calculated with the EMTO method. We also thank Aneta Fronczek for her assistance with many of the illustrations in the paper.

The Center for Atomic-scale Materials Design is funded by the Lundbeck Foundation. We acknowledge support from the Danish Center for Scientific Computation through grant number HDW-0107-07 and from the Danish Council for Strategic Research through grant number 2106-07-0028.

References

[1] International Human Genome Sequencing Consortium 2001 Initial sequencing and analysis of the human genome *Nature* **409** 860
[2] Binnig G, Rohrer H, Gerber C and Weibel E 1982 Surface studies by scanning tunneling microscopy *Phys. Rev. Lett.* **49** 57
[3] Spence J C H and Cowley J M 1978 Lattice imaging in stem *Optik* **50** 129
[4] Moore G E 1965 Cramming more components onto integrated circuits *Electronics* **38** 114
[5] Hohenberg P and Kohn W 1964 Inhomogeneous electron gas *Phys. Rev.* **136** B864
[6] Kohn W and Sham L J 1965 Self-consistent equations including exchange and correlation effects *Phys. Rev.* **140** A1133
[7] Nørskov J K, Scheffler M and Toulhoat H 2006 Density functional theory in surface science and heterogeneous catalysis *MRS Bull.* **31** 669
[8] Holmblad P M, Larsen J H, Chorkendorff I, Nielsen L P, Besenbacher F, Stensgaard I, Lægsgaard E, Kratzer P and Nørskov J K 1996 Designing surface alloys with specific active sites *Catal. Lett.* **40** 131
[9] Besenbacher F, Chorkendorff I, Clausen B S, Hammer B, Molenbroek A M, Nørskov J K and Stensgaard I 1998 Design of a surface alloy catalyst for steam reforming *Science* **279** 1913
[10] Richards W G (ed) 1989 *Computer-Aided Molecular Design* (London: IBC Press)
[11] Walters W P, Stahl M T and Murcko M A 1998 Virtual screening—an overview *Drug Discov. Today* **3** 160
[12] Glick M, Grant G H and Richards W G 2002 Pinpointing anthrax-toxin inhibitors *Nat. Biotechnol.* **20** 118–9
[13] Greeley J and Mavrikakis M 2004 Alloy catalysts designed from first principles *Nat. Mater.* **3** 810
[14] Logadottir A and Nørskov J K 2003 Ammonia synthesis over a Ru(0001) surface studied by density functional calculations *J. Catal.* **220** 273
[15] Honkala K, Hellman A, Remediakis I N, Logadottir A, Carlsson A, Dahl S, Christensen C H and Nørskov J K 2005 Ammonia synthesis from first-principles calculations *Science* **307** 555
[16] Hellman A *et al* 2006 Predicting catalysis: understanding ammonia synthesis from first-principles calculations *J. Phys. Chem.* B **110** 17719
[17] Bligaard T, Nørskov J K, Dahl S, Matthiesen J, Christensen C H and Sehested J 2004 The Brønsted–Evans–Polanyi relation and the volcano curve in heterogeneous catalysis *J. Catal.* **224** 206
[18] Linic S, Jankowiak J and Barteau M A 2004 Selectivity driven design of bimetallic ethylene epoxidation catalysts from first principles *J. Catal.* **224** 489
[19] Nørskov J K, Rossmeisl J, Logadottir A, Lindqvist L, Kitchin J R, Bligaard T and Jonsson H 2004 The origin of the overpotential for oxygen reduction at a fuel cell cathode *J. Phys. Chem.* B **108** 17886
[20] Lopez N, Janssen T V W, Clausen B S, Xu Y, Mavrikakis M, Bligaard T and Nørskov J K 2004 On the origin of the catalytic activity of gold nanoparticles for low temperature co oxidation *J. Catal.* **223** 232
[21] Falsig H, Hvolbæk B, Kristensen I S, Jiang T, Bligaard T, Christensen C H and Nørskov J K 2008 Trends in the catalytic co oxidation activity of nanoparticles *Angew. Chem. Int. Ed. Engl.* **47** 4835
[22] Nørskov J K, Bligaard T, Logadottir A, Kitchin J R, Chen J G, Pandelov S and Stimming U 2005 Trends in the exchange current for hydrogen evolution *J. Electrochem. Soc.* **152** J23–6
[23] Kandoi S, Greeley J, Sanchez-Castillo M A, Evans S T, Gokhale A A, Dumesic J A and Mavrikakis M 2006 Prediction of experimental methanol decomposition rates on platinum from first principles *Top. Catal.* **37** 17
[24] Ceder G, Chiang Y M, Sadoway D R, Aydinol M K, Jang Y I and Huang B 1998 Identification of cathode materials for lithium batteries guided by first-principles calculations *Nature* **392** 694
[25] Franceschetti A and Zunger A 1999 The inverse band-structure problem of finding an atomic configuration with given electronic properties *Nature* **402** 60
[26] Johannesson G H, Bligaard T, Ruban A V, Skriver H L, Jacobsen K W and Nørskov J K 2002 Combined electronic structure and evolutionary search approach to materials design *Phys. Rev. Lett.* **88** 255506
[27] Curtarolo S, Morgan D, Persson K, Rodgers J and Ceder G 2003 Predicting crystal structures with data mining of quantum calculations *Phys. Rev. Lett.* **91** 135503
[28] Morgan D, Ceder G and Curtarolo S 2005 High-throughput and data mining with *ab initio* methods *Meas. Sci. Technol.* **16** 296
[29] Hafner J, Wolverton C and Ceder G 2006 Toward computational materials design: the impact of density functional theory on materials research *MRS Bull.* **31** 659
[30] Fischer C C, Tibbetts K J, Morgan D and Ceder G 2006 Predicting crystal structure by merging data mining with quantum mechanics *Nat. Mater.* **5** 641

[31] Andersson M P, Bligaard T, Kustov A, Larsen K E, Greeley J, Johannessen T, Christensen C H and Nørskov J K 2006 Toward computational screening in heterogeneous catalysis: pareto-optimal methanation catalysts *J. Catal.* **239** 501
[32] Bligaard T, Andersson M P, Jacobsen K W, Skriver H L, Christensen C H and Nørskov J K 2006 Computational materials design from first principles *MRS Bull.* **31** 986
[33] Greeley J, Jaramillo T F, Bonde J, Chorkendorff I B and Nørskov J K 2006 Computational high-throughput screening of electrocatalytic materialsfor hydrogen evolution *Nat. Mater.* **5** 909
[34] Studt F, Abild-Pedersen F, Bligaard T, Sørensen R Z, Christensen C H and Nørskov J K 2008 Rational catalyst design applied to the selective hydrogenation of acetylene *Science* **320** 1320
[35] Alayoglu S, Nilekar A U, Mavrikakis M and Eichhorn B 2008 Ru-Pt core–shell nanoparticles for preferential oxidation of carbon monoxide in hydrogen *Nat. Mater.* **7** 333
[36] Nørskov J K, Bligaard T, Rossmeisl J and Christensen C H 2009 Towards the computational design of solid catalysts *Nat. Chem.* **1** 37
[37] Brown R D, Varma-O'Brien S and Rogers D 2006 Data pipelines and virtual screening: automating the process *QSAR Comb. Sci.* **25** 1181
[38] Ashby M F (ed) 1999 *Materials Selection in Mechanical Design* (Oxford: Butterworth Heinemann)
[39] Microsoft Encarta Online Encyclopedia2007 http://encarta.msn.com *'Catalysis'*
[40] Abild-Pedersen F, Greeley J, Studt F, Rossmeisl J, Munter T R, Moses P G, Skúlason E, Bligaard T and Nørskov J K 2007 Scaling properties of adsorption energies for hydrogen-containing molecules on transition-metal surfaces *Phys. Rev. Lett.* **99** 016105
[41] Abild-Pedersen F and Andersson M P 2007 Co adsorption energies on metals with correction for high coordination adsorption sites—a density functional study *Surf. Sci.* **601** 1747
[42] Nørskov J K *et al* 2002 Universality in heterogeneous catalysis *J. Catal.* **209** 275
[43] Fernandez E M *et al* 2008 Scaling relations for adsorption energies on transition metal oxide, sulfide and nitride surfaces *Angew. Chem. Int. Ed. Engl.* **47** 4683
[44] Bligaard T, Johannesson G H, Ruban A V, Skriver H L, Jacobsen K W and Nørskov J K 2003 Pareto-optimal alloys *Appl. Phys. Lett.* **83** 4527
[45] Ruban A, Hammer B, Stoltze P, Skriver H L and Nørskov J K 1997 Surface electronic structure and reactivity of transition and noble metals *J. Mol. Catal.* A **115** 421
[46] Mavrikakis M, Hammer B and Nørskov J K 1998 Effect of strain on the reactivity of metal surfaces *Phys. Rev. Lett.* **81** 2819
[47] Greeley J and Nørskov J K 2005 A general scheme for the estimation of oxygen binding energies on binary transition metal surface alloys *Surf. Sci.* **592** 104
[48] Hammer B and Nørskov J K 1995 Electronic factors determining the reactivity of metal surfaces *Surf. Sci.* **343** 211
[49] Andersen O K, Jepsen O and Krier G 1994 *Lectures on Methods of Electronic Structure Calculations* chapter Exact muffin–tin orbital theory (Singapore: World Scientific) pp 63–124
[50] Andersen O K and Saha-Dasgupta T 2000 Muffin–tin orbitals of arbitrary order *Phys. Rev.* B **62** 16219
[51] Vitos L, Skriver H L, Johansson B and Kollar J 2000 Application of the exact muffin–tin orbitals theory: the spherical cell approximation *Comput. Mater. Sci.* **18** 24
[52] Vitos L 2001 Total-energy method based on the exact muffin–tin orbitals theory *Phys. Rev.* B **6401** 014107
[53] The CAMD databases www.camd.dtu.dk/English/Research/Databases.aspx
[54] Andersen O K, Jepsen O and Glotzel D 1985 Canonical description of the band structure of metals *Highlights of Condensed Matter Theory* ed F Bassani, F Fumi and M P Tosi (New York: North-Holland) p 59
[55] Ruban A V and Skriver H L 1999 Calculated surface segregation in transition metal alloys *Comput. Mater. Sci.* **15** 119
[56] Java website http://java.sun.com/
[57] Hamilton G (ed) 1997 *JavaBeans API Specification* version 1.01 (Sun Microsystems)
[58] Apache Derby website http://db.apache.org/derby/
[59] 1999 National Emission Inventory Documentation and Data www.epa.gov/ttn/chief/net/1999inventory.html. (US Environmental Protection Agency, 2001)
[60] Kittel C 1996 *Introduction to Solid State Physics* 7th edn (New York: Wiley)
[61] Weast R C (ed) 1968 *CRC Handbook of Chemistry and Physics* 49th edn (Cleveland: CRC)
[62] Padak B, Brunetti M, Lewis A and Wilcox J 2006 Mercury adsorption on activated carbon *Environ. Prog.* **25** 319
[63] Sabatier P and Senderens J B 1902 New methane synthesis *C. R. Hebd. Seances Acad. Sci.* **134** 514
[64] Goodman D W, Kelley R D, Madey T E and Yates J T 1980 Kinetics of the hydrogenation of co over a single crystal nickel-catalyst *J. Catal.* **63** 226

[65] Zubkov T, Morgan G A, Yates J T, Kuhlert O, Lisowski M, Schillinger R, Fick D and Jansch H J 2003 The effect of atomic steps on adsorption and desorption of co on Ru(109) *Surf. Sci.* **526** 57
[66] van Santen R A, de Koster A and Koerts T 1991 The quantum chemical basis of the Fischer-Tropsch reaction *Catal. Lett.* **7** 1
[67] Mavrikakis M, Baumer M, Freund H J and Nørskov J K 2002 Structure sensitivity of CO dissociation on rh surfaces *Catal. Lett.* **81** 153
[68] Sehested J, Larsen K E, Kustov A L, Frey A M, Johannessen T, Bligaard T, Andersson M P, Nørskov J K and Christensen C H 2007 Discovery of technical methanation catalysts based on computational screening *Top. Catal.* **45** 9
[69] Crystal Lattice Structures Web page *Strukturbericht Designation* Center for Computational Materials Science of the United States Naval Research Laboratory http://cst-www.nrl.navy.mil/lattice/
[70] Pettifor D G 1988 Structure maps for pseudobinary and ternary phases *Mater. Sci. Technol.* **4** 675

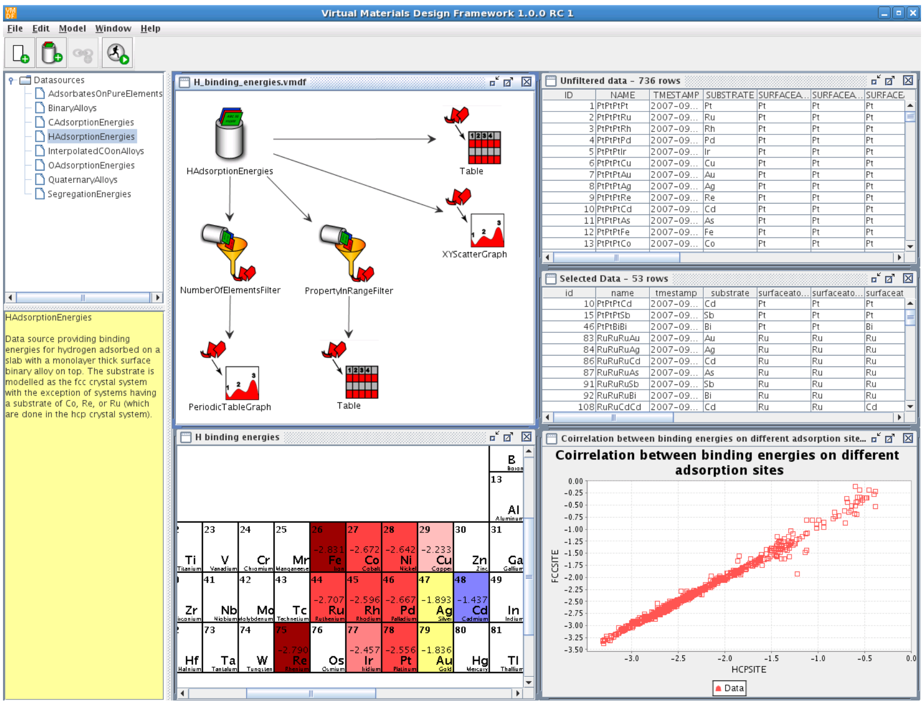
Appendix E

VMDF

The Virtual Materials Design Framework (VMDF) initiated and designed by Ture R. Munter. The new ideas introduced in the VMDF were a complete framework that not only presented data, but was also able to evaluate and query them in a user friendly way.

Collaborations with companies that support the research at CAMD is an important symbioses: the companies need access to early research results and collaborations to implement new methods while the university profits from the exchange of knowledge, the closeness to the market and financially as well. Results are normally exchanged by creating spreadsheets which is a bit cumbersome. To improve the situation results can be put in a database that allows to quickly find things - but requires researchers to know how to write database queries - which they might have known once but to look up a few results they prefer to stay with the spreadsheet. This is where VMDF comes into play: instead of handing over only data the whole visual query enabling framework including the data is exchanged. Visual query execution means that the complexity of a query language is hidden and that all operations are done in a user friendly graphical interface (figure E.1).

Besides the visual queries this framework features different views of the data: tables, plots, checker board plot and even a periodic table view. All views can be exported to different formats as for example png, eps and pdf.



HAdsorptionEnergies

Data source providing binding energies for hydrogen adsorbed on a slab with a monolayer thick surface binary alloy on top. The substrate is modelled as the fcc crystal system with the exception of systems having a substrate of Co, Re, or Ru (which are done in the hcp crystal system).

Unfiltered data - 736 rows

ID	NAME	TMESTAMP	SUBSTRATE	SURFACEA...	SURFACEA...	SURFACE
1	PtPtPtPt	2007-09...	Pt	Pt	Pt	Pt
2	PtPtPtRu	2007-09...	Ru	Pt	Pt	Pt
3	PtPtPtRh	2007-09...	Rh	Pt	Pt	Pt
4	PtPtPtPd	2007-09...	Pd	Pt	Pt	Pt
5	PtPtPtIr	2007-09...	Ir	Pt	Pt	Pt
6	PtPtPtCu	2007-09...	Cu	Pt	Pt	Pt
7	PtPtPtAu	2007-09...	Au	Pt	Pt	Pt
8	PtPtPtAg	2007-09...	Ag	Pt	Pt	Pt
9	PtPtPtRe	2007-09...	Re	Pt	Pt	Pt
10	PtPtPtCd	2007-09...	Cd	Pt	Pt	Pt
11	PtPtPtAs	2007-09...	As	Pt	Pt	Pt
12	PtPtPtFe	2007-09...	Fe	Pt	Pt	Pt
13	PtPtPtCo	2007-09...	Co	Pt	Pt	Pt

Selected Data - 53 rows

id	name	tmestamp	substrate	surfaceato...	surfaceato...	surfaceat
10	PtPtPtCd	2007-09...	Cd	Pt	Pt	Pt
15	PtPtPtSb	2007-09...	Sb	Pt	Pt	Pt
46	PtPtBiBi	2007-09...	Bi	Pt	Pt	Bi
83	RuRuRuAu	2007-09...	Au	Ru	Ru	Ru
84	RuRuRuAg	2007-09...	Ag	Ru	Ru	Ru
86	RuRuRuCd	2007-09...	Cd	Ru	Ru	Ru
87	RuRuRuAs	2007-09...	As	Ru	Ru	Ru
91	RuRuRuSb	2007-09...	Sb	Ru	Ru	Ru
92	RuRuRuBi	2007-09...	Bi	Ru	Ru	Ru
108	RuRuCdCd	2007-09...	Cd	Ru	Ru	Cd

Figure E.1: A screenshot of VMDF. In the top left corner, the available databases are listed. To the right of it is the visual query shown and around it the different output windows. The bottom right windows shows the correlation between the binding of the hcp and fcc adsorption site for H on different surfaces.

Since the whole framework is implemented in Java[100][42] it is possible to run it on almost any operating system.

With VMDF not only the raw data, but also the analysis can be exchanged with the collaborators.

E.1 Work in Progress?

A few new features where added since the first release of VMDF: A filter that is able to join results with a certain criterion, access to external databases and result output to text or csv files. A complete restructuring of the internal data handling and caching reduced memory consumption and made queries faster. A few early access features that can still be improved in stability are also present, like a filter that performs mathematical operations and more complex filters that perform multiple operations at once. Currently there is no ongoing development but the project is available the the CAMD subversion repository.

VMDF is a strong product with many convincing arguments that make it attractive to users - not only that it can be run everywhere but it's also very intuitive. Why was the development paused?

The first reason for pausing was that VMDF did not support collecting data. This means when a user wanted to add new data he/she had to create a spreadsheet and install the development environment or ask a developer to add them. VMDF then had to be re-released to include the new data. Later a new database connector was added to VMDF which allowed to connect to external databases which fixed that shortcoming. Still the amount of data that is in the database now cannot be handled with VMDF. The second reason was that people prefer to use methodologies that they are acquainted with rather than learning something new. Although the programming of the filters is not very difficult, one needs to install the Java Development Kit and know how to program in Java. The third reason is that the user interface code and the serialization code should be made more flexible to cope with future needs as for example the design of recursive filters and to be able to open older versions of the file format used to store the visual query.

The technical shortcomings can be fixed with manpower. The problem of using too much memory for example can be solved by performing prior query-analysis and determining what fields should actually be retrieved or much more efficient translate the query straight into an SQL statement. This would reduce the amount of data enough so that it can be handled by VMDF. The file format for the user-interface can be designed more flexible and future versions would be backwards compatible. Java was a good choice for companies that wanted to profit from WORA (write once, run anywhere) and have people appointed for maintaining and updating a software package. It's much harder to build a community of researchers that work on Java program because of the barriers. Even this problem can be solved by building a meta layer between the users and Java: A filter creator could allow people to put together a few low level filters to create a high level filter.

The technical arguments and the barrier of learning a new programming language can be defeated. The only issue that cannot be fixed is that Java has lost popularity to more flexible script languages that have reached a maturity level that enables applications to run in a web browser without the need to install any software. They run on all platforms and even on cellular phones and use less resources.

E.2 Conclusion

VMDF is a very good product with a big potential, but in order to be able to compete, it needs appointed staff that maintains close contact with the researchers in order to keep the product up to date and attractive. The integrated database that contains pre-filtered and indexed data could be used to sell the data to companies and the other part could be available for free and used to connect to other databases.

VMDF was a framework that enables average users to perform more complex materials design queries visually in a database and display the results in a useful way and enabling exports in different file formats. It is written in Java which enables it to run on almost any platform, but at the same time Java is a barrier because the language is not as popular as it used to be. In order to continue the development of VMDF the internal query engine should be improved, a new meta-level for the creation of new filters should be created an some staff for the maintenance should be appointed. In this way at the product could at least compete with the newer developed interfaces to the database. The SiGUI interface to the database is inspired by the visualization of the VMDF queries and implemented in javascript (see 3.3.1.d.1 SiGUI, a silo plug-in).

Appendix F

Appendix

F.1 Extracted Values from Codes

The following lists shows the parameters that are extracted and stored in the db-files. Note that not all output files contain always all variables. The parameters are ordered by capitalization and then alphabetically.

Dacapo
AtomicNumbers, BZKpoints, ChargeMixing, ConvergenceControl, Date, Density_WaveCutoff, DynamicAtomAttributes, DynamicAtomForces, DynamicAtomPositions, DynamicAtomSpecies, DynamicAtomVelocities, EigenValues, ElectronicBands, ElectronicMinimization, EnsembleXCEnergies, EvalFunctionalOfDensity_XC, EvaluateCorrelationEnergy, EvaluateExchangeEnergy, EvaluateTotalEnergy, ExcFunctional, FermiLevel, IBZKpoints, InitialAtomicMagneticMoment, Keywords, KpointWeight, MagneticMoment, NetCDFOutputControl, OccupationNumbers, PlaneWaveCutoff, StructureFactor, SymmetryGeneratorOrder, TotalEnergy, TotalFreeEnergy, TotalStress, TypeNLProjectorl, TypeNLProjectorm, UnitCell, UseSymmetry, User, ase _atomic_numbers, ase_cell, ase_center_of_mass, ase_charges, ase_chemical_symbols, ase_dipole_moment, ase_forces, ase_initial _magnetic_moments, ase_kinetic_energy, ase_magnetic_moment, ase_magnetic_moments, ase_masses, ase_momenta, ase_moments_of_inertia, ase_name, ase_number_of_atoms, ase_pbc, ase_positions, ase_potential_energy, ase_reciprocal_cell,

ase_scaled_positions, ase_tags, ase_temperature, ase_total_energy, ase_version,
ase_volume, atomdos_angular_channels, atomdos_energygrid_size, atomdos_pro-
jections, atomdos_radial_orbs, hardgrid_dim1, hardgrid_dim2, hardgrid_dim3,
longstring, max_projectors_per_atom, number_BZ_kpoints, number _IBZ_kpoints,
number_ionic_steps, number_of_bands, number_of_dynamic_atoms, number_of_-
spin, number_of_symm_gen, number_plane _waves, real_complex, softgrid_dim1,
softgrid_dim2, softgrid_dim3

Gaussian
Atomic_numbers, Basis_set, Charge, Chemical_formula, Compact_data, Com-
puter_system, Date, HF, Method, Multiplicity, Person, Positions, Sequence_num-
ber, Title, Type_of_run

GPAW
ActiniumFingerprint, AluminiumFingerprint, AmericiumFingerprint, AntimonyFin-
gerprint, ArgonFingerprint, ArsenicFingerprint, AstatineFingerprint, Atomic-
Numbers, BZKPoints, BariumFingerprint, BasisSet, BerkeliumFingerprint, Beryl-
liumFingerprint, BismuthFingerprint, BoronFingerprint, BoundaryConditions,
BromineFingerprint, CadmiumFingerprint, CaesiumFingerprint, CalciumFinger-
print, CaliforniumFingerprint, CarbonFingerprint, CartesianForces, Cartesian-
Positions, CeriumFingerprint, Charge, ChlorineFingerprint, ChromiumFinger-
print, CobaltFingerprint, Converged, CopperFingerprint, CuriumFingerprint,
DataType, DensityConvergenceCriterion, DensityError, DysprosiumFingerprint,
Ebar, Eext, EigenstateError, EigenstatesConvergenceCriterion, Eigenvalues, Ein-
steiniumFingerprint, Ekin, EnergyConvergenceCriterion, EnergyError, Epot,
ErbiumFingerprint, EuropiumFingerprint, Exc, FermiLevel, FermiWidth, Fer-
miumFingerprint, FixDensity, FixMagneticMoment, FluorineFingerprint, For-
Transport, FranciumFingerprint, GadoliniumFingerprint, GalliumFingerprint,
GermaniumFingerprint, GoldFingerprint, GridSpacing, HafniumFingerprint, He-
liumFingerprint, HolmiumFingerprint, HydrogenFingerprint, IBZKPointWeights,
IBZKPoints, IndiumFingerprint, InterpolationStencil, IodineFingerprint, Iridi-
umFingerprint, IronFingerprint, KohnShamStencil, KryptonFingerprint, Lan-
thanumFingerprint, LawrenciumFingerprint, LeadFingerprint, LithiumFinger-
print, LutetiumFingerprint, MagnesiumFingerprint, MagneticMoments, Man-
ganeseFingerprint, MaximumAngularMomentum, MendeleviumFingerprint, Mer-
curyFingerprint, MixBeta, MixClass, MixMetric, MixOld, MixWeight, Mode,
MolybdenumFingerprint, NeodymiumFingerprint, NeonFingerprint, Neptuni-
umFingerprint, NickelFingerprint, NiobiumFingerprint, NitrogenFingerprint,
NobeliumFingerprint, NumberOfBandsToConverge, OccupationNumbers, Osmi-
umFingerprint, OxygenFingerprint, PalladiumFingerprint, PhosphorusFinger-
print, PlatinumFingerprint, PlutoniumFingerprint, PoissonStencil, PoloniumFin-
gerprint, PotassiumFingerprint, PotentialEnergy, PraseodymiumFingerprint,

Promethium-Fingerprint, ProtactiniumFingerprint, RadiumFingerprint, Radon-Fingerprint, RheniumFingerprint, RhodiumFingerprint, RubidiumFingerprint, RutheniumFingerprint, S, SamariumFingerprint, ScandiumFingerprint, SeleniumFingerprint, SetupTypes, SiliconFingerprint, SilverFingerprint, SodiumFingerprint, SoftGauss, StrontiumFingerprint, SulfurFingerprint, Tags, TantalumFingerprint, TechnetiumFingerprint, TelluriumFingerprint, TerbiumFingerprint, ThalliumFingerprint, ThoriumFingerprint, ThuliumFingerprint, Time, TinFingerprint, TitaniumFingerprint, TungstenFingerprint, UnitCell, UnnilhexiumFingerprint, UnnilpentiumFingerprint, UnnilquadiumFingerprint, UraniumFingerprint, UseSymmetry, VanadiumFingerprint, XCFunctional, XenonFingerprint, YtterbiumFingerprint, YttriumFingerprint, ZincFingerprint, ZirconiumFingerprint, architecture, ase_atomic_numbers, ase_cell, ase_center_of_mass, ase_charges, ase_chemical _symbols, ase_dipole_moment, ase_dir, ase_forces, ase_initial_magnetic_moments, ase_kinetic_energy, ase_magnetic_moment, ase _magnetic_moments, ase_masses, ase_momenta, ase_moments_of_inertia, ase_name, ase_number_of_atoms, ase_pbc, ase_positions, ase _potential_energy, ase_reciprocal_cell, ase_scaled_positions, ase_tags, ase_temperature, ase_total_energy, ase_version, ase_volume, energyunit, gpaw_dir, history, lengthunit, nadm, natoms, nbands, nbzkpts, nfinegptsx, nfinegptsy, nfinegptsz, ngptsx, ngptsy, ngptsz, nibzkpts, norbitals, nproj, nspins, pid, pid, units, version

VASP
EDIFF, EDIFFG, EMAX, EMIN, ENAUG, ENMAX, IALGO, IBRION, ICHARG, ISIF, ISMEAR, ISPIN, ISTART, LORBIT, MAGMOM, NBANDS, NELECT, NELM, NELMDL, NELMIN, NPAR, NSW, POTIM, PREC, SIGMA, SYSTEM, VOSKOWN, atomic_numbers, atoms, cellf, date, divisions, e_0_energy, e_fr_energy, e_wo _entrp, elementnames, forces, kscheme, location, mass, platform, program, pseudo, stress, subversion, types, user, valence, volf, voli

F.2 PHPUI script to continue analysis in the PUI

This section presents a script downloaded a CMR web server from the PHP/HTML interface. The query in the web-interface was `db_user=ivca atoms=Cu ABO3` and the same query is run when the script is.

```
# This data was downloaded from CMR
# Link to whole dataset: http://cmr.fysik.dtu.dk/...
# ****************************************
# If you use data from others don't forget to cite!
# ****************************************

from cmr.ui import DBReader
from cmr.tools.type_converters import DateConverterBase

def date_convert(string):
    try:
        return DateConverterBase.convert(string)
    except:
        return string


connection_name="default"
reader = DBReader(connection_name, db_prefix="ivca")

# Converting atomic names to atomic numbers
from cmr.static import atomic_numbers
atom_name_list=["Cu"]
atoms = [atomic_numbers[atom_name] for atom_name in atom_name_list]

# add here the columns that should be downloaded and shown:
# please note that you should never use id_ref - use db_hash instead
columns = ["db_calculator"]
collection = reader.find(keyword_list=["ABO3"],
                         atom_list=atoms,
                         name_op_value_list=[('db_user', '=', 'ivca')],
                         columns=columns)

collection.print_table(columns=columns)

################
#  Add keywords and fields to selected items and submit
#  it to the database again
################
# add keywords and other fields:
# result_dict = collection.get_as_dict(columns=["db_hash"])
# hash_list = result_dict["db_hash"]
# for hash in hash_list:
#      data = reader.retrieve(hash)
#      #add a keyword to all
#      data.get("db_keywords", []).append("my_new_keyword")
#      #add a user_defined field verified
#      data.set_user_variable("verified", True)
#      #write it to the database
#      data.write(".db")

#################
#  CREATE a group with seleced item
#################
# result_dict = collection.get_as_dict(columns=["db_hash"])
# hash_list = result_dict["db_hash"]
# collection.create_group(cmr_params={"db_keywords":["new group"]})

#################
#  DELETE selected data from database
#################
# import cmr
# result_dict = collection.get_as_dict(columns=["db_hash"])
# hash_list = result_dict["db_hash"]
# cmr.delete(hash_list)
```

F.3 Deployment Examples and Dependencies

Depending on the available resources and the size of the project, CMR can be installed differently. Three examples are going to be presented.

In order to create db-files, often the original calculator and all its dependencies must be installed as well. Currently this applies to Dacapo, GPAW and ASE trajectory files.

Here is a list with the explanation what the comonents are for. More information on how to install these components can be found in the CMR wiki `https://wiki.fysik.dtu.dk/cmr` in the *Server Setup* section.

- Python 2.4.3 or newer
  The Python interpreter is needed because CMR is implemented in Python
- (python-simplejson extension - included in Python 2.6 and newer)
  When db-files are uploaded to the database also a representation in json is created that is retrieved upon querying and needs to be parsed - which is slower if Python-simplejson is not installed
- CMR
  CMR can be installed from the repository or from the RPMS created by Marcin Dulak. More information can be found in the CMR wiki.
- CMR database upload scripts, available from the CMR wiki
  Scripts that check the db-file repository for new files, perform the validation and the upload to the database
- MySQL 5.0.77
  The MySQL database - for user systems the MySQL client is enough, for server the MySQL server needs to be installed.
- MySQL-python or PyMySQLPyMySQL[101]
  Python cannot natively connect to the MySQL database and needs one of the above extensions to do so.
- Apache server (httpd)
  The webserver.
- PHP
  Needed to create the web-pages.
- PHP json extension
  Needed to parse the json representation.
- cron
  If the data should be automatically uploaded to the database a *cron job* executes the upload script in a certain interval.
- xmllint
  Xmllint is used to perform the XML validation. CMR can also validate db-files without it, but this is not automatically supported, therefore it is

required when uploading data to the database

F.3.1 Minimal

A minimal environment can handle up to a couple of thousand calculations. The limiting factor is the time it takes to read the db-files or the db-file cache from the repository to memory and may vary from machine to machine. The simplejson extension is optional but strongly recommended to install .

- Python 2.4.3 or newer
- (python-simplejson - included in Python 2.6 and newer)
- CMR
- (xmllint - if files should be XML-validated)

F.3.2 Medium

A medium size environment can handle many calculations as long as the query is performed by the database; the bottleneck is the download of the db-files from the MySQL database. Because of the redundancy in the MySQL database of the uploaded data there is considerable amount of extra disk space needed - approximately a factor five of the disk size of the db-files.

- Python 2.4.3 or newer
- (python-simplejson - included in Python 2.6 and newer)
- CMR
- CMR database upload scripts, available from the CMR wiki
- MySQL 5.0.77
- MySQL-python or PyMySQLPyMySQL[101]
- (cron - if data should be automatically uploaded to the MySQL database)
- xmllint

F.3.3 Institute or high quality database

The deployment for an institute or for the high quality database is the same.

- Python 2.4.3 or newer
- (python-simplejson extension - included in Python 2.6 and newer)
- CMR

- CMR database upload scripts, available from the CMR wiki
- MySQL 5.0.77
- MySQL-python or PyMySQL
- Apache server (httpd)
- PHP
- PHP json extension
- cron

F.4 GA minimization script template

The example in section 4.2.5 shows a partial script template used in the GA to minimize a function. The script template is used to create a scripts for every allele that is then updated with the fitness upon termination. Listing F.1 shows the full script template for the discussed example. The special tags `#<XXX_START>` and `#<XXX_END>` are used to mark sections of the script. All content between `#<FILL_START>` and `#<FILL_END>` is for example replaced with actual information about the allele in our case with the coordinate, the parents, the creator, the hashes, where to write the result to et cetera. The information that is shown in the template is actually real data. It is accessed by the `run` method (within `#<RUN_START>` and `#<RUN_END>`) and must be customized by the user. `run` determines the fitness, and writes a db-file with the result (`write_spreadsheet`). This result db-file is added as a member to the allele (`add_to_group`), so that it is possible to trace where the data came from. If the calculation of the fitness had been a GPAW calculation, we would have used `write_gpaw_result` instead of `write_spreadsheet`. Finally the functions `get_info` `print_info` get and print information about the state of the allele. This can be used for debugging or checking the status without actually running the GA.

```
import os
import sys
import cmr
from cmr.base.converter import Converter
import random

#<FILL_START>
# ALL CONTENT HERE WILL AUTOMATICALLY BE REPLACED
# EVERYTHING IN parameters IS ALSO WRITTEN TO THE
# OUTPUT AND THE GROUP FILES!
parameters = {}
parameters['db_scripts'] = ['8.55741e02_8.48785e02.py']
parameters['parents_oids'] = [29]
parameters['parents_names'] = ['9.50768e02_7.82576e02']
parameters['generation'] = 5
parameters['group_db_file'] = '8.55741e02_8.48785e02_group.db'
parameters['oid'] = 39
parameters['db'] = False
parameters['coord'] = [-855.74062349082124, 848.78462159332571]
parameters['before_hash'] = '...'
parameters['ga_hash'] = '...'
parameters['ga_name'] = '...'
parameters['parents'] = ['...']
parameters['calc_db_file'] = '8.55741e02_8.48785e02_calc.db'
parameters['creator'] = 'vectors.MutationX(-2.00e+02, 2.00e+02)'
parameters['after_hash'] = '...''
#<FILL_END>
```

```
def add_to_group(cmr_params):
    """adds a reference to the group to the given calculation
       and update the fitness value.
    """
    group = cmr.read(parameters["group_db_file"])
    data = cmr.read(parameters["calc_db_file"])
    group.add(data.get_hash())
    p = cmr_params.copy()
    p["output"] = parameters["group_db_file"]
    p["state"] = "successful"
    group.write(p) # write this group file


def write_gpaw_result(calc, cmr_params):
    if parameters.has_key("gpaw_output"):
        calc.write(parameters["gpaw_output"])
    calc.write(parameters["calc_db_file"], cmr_params=cmr_params)
    add_to_group(cmr_params)

def write_spreadsheet(cmr_params):
    from cmr.static import CALCULATOR_SPREADSHEET
    data = Converter.get_xml_writer(CALCULATOR_SPREADSHEET)
    for key, value in cmr_params.items():
        data.set_user_variable(key, value)
    data.write(parameters["calc_db_file"], db=parameters["db"])
    add_to_group(cmr_params)


#<RUN_START>
def run():
    from ga_examples.fe_ext_calc.functions import function1

    if random.randint(0, 10) == 1:
        raise Exception("Random failure")
    result = function1(parameters["coord"][0], parameters["coord"][1])

    cmr_params = dict(parameters.items())
    cmr_params["fitness"] = result
    #add more results here, if desired:
    #cmr_params["..."] = ...

    write_spreadsheet(cmr_params)
#<RUN_END>

def get_info():
    """returns the information about this allele"""
    info = {}
    for k, v in parameters.items():
        info[k] = v
    #check, if script has finished
    if not os.path.exists(parameters["calc_db_file"]):
        (file_base, ext) = os.path.splitext(__file__)
        err_file = file_base + ".err"
        queued = file_base + ".py.queued"
        if os.path.exists(err_file):
            if os.stat(err_file).st_size > 0:
                info["state"] = "execution failed"
            else:
                info["state"] = "running"
        elif os.path.exists(queued):
                info["state"] = "queued"
        else:
            info["state"] = "not queued"
    else:
        info["state"] = "done"
    return info

def print_info():
    """prints all static information about this script
    algorithm as a python dictionary
    """
    print get_info().__repr__()

if __name__ == "__main__":
    if len(sys.argv) > 1 and sys.argv[1] == "-info":
        print_info()
    elif len(sys.argv) > 1:
        print "Unknown arguments", sys.argv[1:]
    else:
        run()
```

Listing F.1: `script_template.py`

F.5 Inside a db-file

A minimal db-file is a compressed tar.bz2 container containing a file named ”info.xml”. ”info.xml” contains the data of either a calculation or the definition for a group of calculations. Figure F.1 shows an example of XML code to demonstrate how the data is stored in ”info.xml”:

```
 ...
 <calculator>
   <TotalEnergy><double>-586.442477162</double>
   </TotalEnergy>
   <AtomicNumbers>
    <long_array length="2">
     <long>6</long>
     <long>8</long>
    </long_array>
   </AtomicNumbers>
...
</calculator>
 ...
```

Figure F.1: XML extract from a GPAW calculation as it is stored within ”info.xml” in a db-file. The content is as follows: $TotalEnergy = -586.442477162$ and $AtomicNumbers = [6, 8]$. ”double”, ”long_array” and ”long” are the names of the internal type.

The XML-file is divided into seven paragraphs as illustrated in F.5:

- **static:** data that is present in every db-file like the date and the user name
- **extras:** information that describe the calculation like keywords and a description
- **calculator:** all data from the original calculator output that was selected to go into the db-file
- **user:** custom data that the user can chosen freely
- **ASE:** data from retrieved from the ASE interface; only available, if ASE is used with this file format; this will contain some duplicate data that is already present in the calculator paragraph - but the units will be eV and the coordinate absolute and not relative.
- **history:** when db-files are modified and the hash value changes the previous hash value should be kept as a backup; in the future there might be ability to search also for older hash values

- **runtime:** information about how the calculation was run. Currently none of the converter collects this information automatically

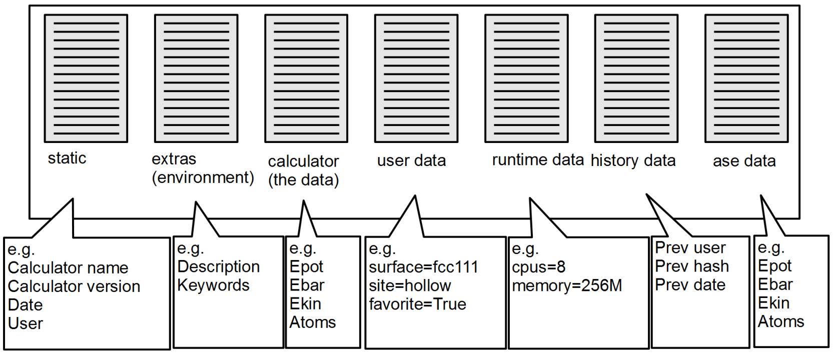

static
extras (environment)
calculator (the data)
user data
runtime data
history data
ase data
e.g. Calculator name Calculator version Date User
e.g. Description Keywords
e.g. Epot Ebar Ekin Atoms
e.g. surface=fcc111 site=hollow favorite=True
e.g. cpus=8 memory=256M
Prev user Prev hash Prev date
e.g. Epot Ebar Ekin Atoms

Figure F.2: Detailed content of a db-file. The shown variable names are of descriptive nature to show what goes where. The ASE data seems to be duplicate - but it actually presents the data in a way that the users are used to see it in terms of units and absolute coordiantes.

Bibliography

[1] Alec Belsky, Mariette Hellenbrandt, Vicky Lynn Karen, and Peter Luksch. New developments in the Inorganic Crystal Structure Database (ICSD): accessibility in support of materials research and design. *Acta Crystallogr., Sect. B*, 58(3 Part 1):364–369, Jun 2002.

[2] Inorganic Crystal Structure Database (ICSD). `http://www.fiz-karlsruhe.de`, 2012-01-31.

[3] AflowLib: Ab-initio Electronic Structure Library. `http://www.aflowlib.org/`, 2012-01-31.

[4] National Center for Biotechnical Information (NCBI). `http://www.ncbi.nlm.nih.gov/`, 2012-01-31.

[5] T Hubbard, D Barker, E Birney, G Cameron, Y Chen, L Clark, T Cox, J Cuff, V Curwen, T Down, R Durbin, E Eyras, J Gilbert, M Hammond, L Huminiecki, A Kasprzyk, H Lehvaslaiho, P Lijnzaad, C Melsopp, E Mongin, R Pettett, M Pocock, S Potter, A Rust, E Schmidt, S Searle, G Slater, J Smith, W Spooner, A Stabenau, J Stalker, E Stupka, A Ureta-Vidal, I Vastrik, and M Clamp. The Ensembl genome database project. *Nucleic Acids Res.* , 30(1):38–41, JAN 1 2002.

[6] Protein Data Bank (PDB). `http://www.pdb.org/`, 2012-01-31.

[7] Human Genome Project. `http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml`, , 2012-01-31.

[8] CAMd Database. `http://www.camd.dtu.dk/Forskning/Databases.aspx`, 2012-01-31.

[9] Gary Yuan and François Gygi. ESTEST: a framework for the validation and verification of electronic structure codes. *Comput. Sci. Discovery*, 3(1):015004, 2010.

[10] Anubhav Jain, Geoffroy Hautier, Charles J. Moore, Shyue Ping Ong, Christopher C. Fischer, Tim Mueller, Kristin A. Persson, and Gerbrand Ceder. A high-throughput infrastructure for density functional theory calculations. *Comp. Mater. Sci*, 50(8):2295 – 2310, 2011.

[11] MaterialsGenome. `http://www.materialsgenome.org/`, 2012-01-31.

[12] Quixote. `http://quixote.wikispot.org/`, 2012-01-31.

[13] Sam Adams, Pablo de Castro, Pablo Echenique, Jorge Estrada, Marcus Hanwell, Peter Murray-Rust, Paul Sherwood, Jens Thomas, and Joe Townsend. The Quixote project: Collaborative and Open Quantum Chemistry data management in the Internet age. *Journal of Cheminformatics*, 3(1):38, 2011.

[14] Quantum Materials Informatics Project. `http://www.qmip.org/`.

[15] S. R. Bahn and K. W. Jacobsen. An object-oriented scripting interface to a legacy electronic structure code. *Comput. Sci. Eng.*, 4(3):56–66, may 2002.

[16] Python. `http://python.org/`, 2012-01-31.

[17] Comma-separated values. `http://en.wikipedia.org/wiki/Comma-separated_values`, 2012-01-31.

[18] F.P. Miller, A.F. Vandome, and J. McBrewster. *JSON: Computer, Human-readable Medium, Data Structure, Associative Array, Douglas Crockford, Internet Media Type, Serialization, Ajax (programming), XML, JavaScript, Ecma International.* VDM Publishing House Ltd., 2009.

[19] XYZ file format. `http://openbabel.sourceforge.net/wiki/XYZ`, 2012-01-31.

[20] B. Hammer, L. B. Hansen, and J. K. Nørskov. Improved adsorption energetics within density-functional theory using revised Perdew-Burke-Ernzerhof functionals. *Phys. Rev. B*, 59(11):7413–7421, Mar 1999.

[21] J. J. Mortensen, L. B. Hansen, and K. W. Jacobsen. Real-space grid implementation of the projector augmented wave method. *Phys. Rev. B*, 71:035109, Jan 2005.

[22] G. Kresse and J. Furthmüller. Efficiency of ab-initio total energy calculations for metals and semiconductors using a plane-wave basis set. *Comp. Mater. Sci*, 6(1):15 – 50, 1996.

[23] R.M. Martin. *Electronic structure: basic theory and practical methods.* Cambridge University Press, 2004.

[24] Atomic Simulation Environment (ASE). `https://wiki.fysik.dtu.dk/ase/`, 2012-01-31.

[25] A.T. Holdener. *Ajax: the definitive guide.* Definitive Guide Series. O'Reilly, 2008.

[26] C. Snyder, T. Myer, and M. Southwell. *Pro PHP Security: From Application Security Principles to the Implementation of XSS Defenses.* Apress Series. Apress, 2010.

[27] M. Nystrom. *SQL Injection Defenses.* O'Reilly shortcuts. O'Reilly, 2007.

[28] Peter Pin shan Chen. The Entity-Relationship Model: Toward a Unified View of Data. *ACM T. Database Syst.*, 1:9–36, 1976.

[29] *MySQL 5.0 Reference Manual.* 01 2012.

[30] Nick Jennings and Michael Wooldridge. Software Agents. In *IEE Review*, pages 17–20, 1996.

[31] Cost of Hard Drive Storage Space. `http://ns1758.ca/winch/winchest.html`, 2012-01-31.

[32] XML Technology. `http://www.w3.org/standards/xml/`, 2012-01-31.

[33] Extensible Markup Language (XML) 1.1 (Second Edition). `http://www.w3.org/TR/2006/REC-xml11-20060816/`, 2012-01-31.

[34] Tar. `http://www.gnu.org/software/tar/`, 2012-01-31.

[35] bzip2. `http://bzip.org/`, 2012-01-31.

[36] Xavier Gonze, C.-O. Almbladh, A. Cucca, D. Caliste, C. Freysoldt, M. A. L. Marques, V. Olevano, Yann Pouillon, and M. J. Verstraete. Specification of an extensible and portable file format for electronic structure and crystallographic data. *CoRR*, abs/0805.0192, 2008.

[37] R. Rew and G. Davis. NetCDF: an interface for scientific data access. *IEEE Comput. Graphics Appl.*, 10(4):76–82, jul 1990.

[38] NetCDF. `http://www.unidata.ucar.edu/software/netcdf/`, 2012-01-31.

[39] M.R. Overly, Pike, and Inc Fischer. *The open source handbook.* Pike & Fischer, 2003.

[40] A. Tveito, H.P. Langtangen, B.F. Nielsen, and X. Cai. *Elements of Scientific Computing.* Texts in Computational Science and Engineering. Springer, 2010.

[41] Mathieu Fourment and Michael Gillings. A comparison of common programming languages used in bioinformatics. *BMC Bioinf.*, 9(1):82, 2008.

[42] H. Schildt. *Java The Complete Reference, 8th Edition.* The Complete Reference. McGraw-Hill Companies,Inc., 2011.

[43] M. Gregoire, N.A. Solter, and S.J. Kleper. *Professional C++.* John Wiley & Sons, 2011.

[44] Fernando Perez, Brian E. Granger, and John D. Hunter. Python: An Ecosystem for Scientific Computing. *Comput. Sci. Eng.*, 13:13–21, 2011.

[45] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13:377–387, June 1970.

[46] S. Tiwari. *Professional NoSQL.* John Wiley & Sons, 2011.

[47] S.W. Dietrich and S.D. Urban. *Fundamentals of Object Databases: Object-Oriented and Object-Relational Design.* Synthesis Lectures on Data Management. Morgan & Claypool, 2011.

[48] Rick Cattell. Scalable SQL and NoSQL data stores. *SIGMOD Rec.*, 39:12–27, May 2011.

[49] M. Doyle. *Beginning PHP 5.3.* John Wiley & Sons, 2011.

[50] J. Gerner, E. Naramore, M. Owens, and M. Warden. *Professional LAMP: Linux, Apache, MySQL and PHP5 Web Development.* John Wiley & Sons, 2006.

[51] T. Butzon. *PHP by example.* By Example. Que Pub., 2002.

[52] Web Server Survey. `http://news.netcraft.com/survey/`, 2012-01-31.

[53] T. Tomlinson and J. VanDyk. *Pro Drupal 7 Development, Third Edition.* Apress Series. Apress, 2010.

[54] The Apache Software Foundation. *Apache HTTP Server 2.2 Official Documentation - Volume III. Modules (A-H).* Fultus Corporation, 2010.

[55] Helen M. Berman, John D. Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The Protein Data Bank. *Nucleic Acids Res.*, 28(1):235–242, 2000.

[56] NCBI. `http://www.ncbi.nlm.nih.gov/`, 2012-01-31.

[57] Akihiko Kudo and Yugo Miseki. Heterogeneous photocatalyst materials for water splitting. *Chem. Soc. Rev.*, 38:253–278, 2009.

[58] Xiaobo Chen, Shaohua Shen, Liejin Guo, and Samuel S. Mao. Semiconductor-based Photocatalytic Hydrogen Generation. *Chem. Rev.*, 110(11):6503–6570, 2010.

[59] Oleg Gritsenko, Robert van Leeuwen, Erik van Lenthe, and Evert Jan Baerends. Self-consistent approximation to the Kohn-Sham exchange potential. *Phys. Rev. A*, 51:1944–1954, Mar 1995.

[60] M. Kuisma, J. Ojanen, J. Enkovaara, and T. T. Rantala. Kohn-Sham potential with discontinuity for band gap materials. *Phys. Rev. B*, 82:115106, Sep 2010.

[61] Hiroshi Mizoguchi, Hank W Eng, and Patrick M Woodward. Probing the electronic structures of ternary perovskite and pyrochlore oxides containing Sn(4+) or Sb(5+). *Inorg. Chem.*, 43(5):1667–1680, March 2004.

[62] Hideki Kato, Hisayoshi Kobayashi, and Akihiko Kudo. Role of Ag+ in the Band Structures and Photocatalytic Properties of AgMO3 (M: Ta and Nb) with the Perovskite Structure. *J. Phys. Chem. B*, 106(48):12441–12447, 2002.

[63] D Yamasita, T Takata, M Hara, JN Kondo, and Kazunari Domen. Recent progress of visible-light-driven heterogeneous photocatalysts for overall water splitting. *Solid State Ionics*, 172:591–595, 2004.

[64] Huashun Zhang, Yaogang Li, Qinghong Zhang, and Hongzhi Wang. Preparation of high surface area LaTiO2N photocatalyst. *Mater. Lett.*, (17-18):2729–2732, June 2008.

[65] pulp-or. `http://code.google.com/p/pulp-or/`, 2012-01-31.

[66] John H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992.

[67] Web of Knowledge. `http://wokinfo.com/`, 2012-01-31.

[68] Yehuda Zeiri. Prediction of the lowest energy structure of clusters using a genetic algorithm. *Phys. Rev. E*, 51:R2769–R2772, Apr 1995.

[69] D. M. Deaven and K. M. Ho. Molecular Geometry Optimization with a Genetic Algorithm. *Phys. Rev. Lett.*, 75(2):288–291, Jul 1995.

[70] G. H. Jóhannesson, T. Bligaard, A. V. Ruban, H. L. Skriver, K. W. Jacobsen, and J. K. Nørskov. Combined Electronic Structure and Evolutionary Search Approach to Materials Design. *Phys. Rev. Lett.*, 88:255506, Jun 2002.

[71] D.G. Mayer. *Evolutionary algorithms and agricultural systems.* The Kluwer international series in engineering and computer science. Kluwer Academic Publishers, 2002.

[72] JM Pena, V Robles, P Larranaga, V Herves, F Rosales, and MS Perez. GA-EDA: Hybrid evolutionary algorithm using genetic and estimation of distribution algorithms. In Orchard, B and Yang, C and Ali, M, editor, *Innovations in applied Artificial Intelligence*, volume 3029 of *Lecture notes in computer science*, pages 361–371, 2004.

[73] Michel Gendreau and Jean-Yves Potvin. *Handbook of Metaheuristics.* Springer; 2nd ed. edition (September 30, 2010), 2010.

[74] M. Fowler. *UML distilled: a brief guide to the standard object modeling language.* Addison-Wesley object technology series. Addison-Wesley, 2004.

[75] E. Gamma. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley professional computing series. Addison-Wesley, 1995.

[76] A. M. Molenbroek, S. Haukka, and B. S. Clausen. Alloying in Cu/Pd Nanoparticle Catalysts. *J. Phys. Chem. B*, 102(52):10680–10689, 1998.

[77] Min-Hua Shao, Kotaro Sasaki, and Radoslav R. Adzic. PdFe Nanoparticles as Electrocatalysts for Oxygen Reduction. *J. Amer. Chem. Soc.*, 128(11):3526–3527, 2006. PMID: 16536519.

[78] Asbjorn Klerke, Claus Hviid Christensen, Jens K. Norskov, and Tejs Vegge. Ammonia for hydrogen storage: challenges and opportunities. *J. Mater. Chem.*, 18:2304–2310, 2008.

[79] Egill Skulason, Thomas Bligaard, Sigridur Gudmundsdottir, Felix Studt, Jan Rossmeisl, Frank Abild-Pedersen, Tejs Vegge, Hannes Jonsson, and Jens K. Norskov. A theoretical evaluation of possible transition metal electro-catalysts for N2 reduction. *Phys. Chem. Chem. Phys.*, 14:1235–1245, 2012.

[80] Wonjae Lee and Hak-Young Kim. Genetic algorithm implementation in Python. In *Computer and Information Science, 2005. Fourth Annual ACIS International Conference on*, pages 8 – 11, 2005.

[81] Pyevolve 0.5. `http://pypi.python.org/pypi/Pyevolve`, 2012-01-31.

[82] Distributed Evolutionary Algorithms in Python (DEAP) 0.6. `http://code.google.com/p/deap/`, 2012-01-31.

[83] Shin-ichi Orimo, Yuko Nakamori, Jennifer R. Eliseo, Andreas Züttel, and Craig M. Jensen. Complex Hydrides for Hydrogen Storage. *Chem. Rev.*, 107(10):4111–4132, 2007.

[84] A. Züttel, S. Rentsch, P. Fischer, P. Wenger, P. Sudan, Ph. Mauron, and Ch. Emmenegger. Hydrogen storage properties of LiBH4. *J. Alloys Compd.*, 356-357(0):515–520, 2003. Proceedings of the Eighth International Symposium on Metal-Hydrogen Systems, Fundamentals and Applications (MH2002).

[85] Zbigniew Łodziana and Tejs Vegge. Structural Stability of Complex Hydrides: $LiBH_4$ Revisited. *Phys. Rev. Lett.*, 93:145501, Sep 2004.

[86] Zbigniew Łodziana and Tejs Vegge. Łodziana and Vegge Reply:. *Phys. Rev. Lett.*, 97:119602, Sep 2006.

[87] Krzysztof Chlopek, Christoph Frommen, Aline Leon, Oleg Zabara, and Maximilian Fichtner. Synthesis and properties of magnesium tetrahydroborate, Mg(BH4)2. *J. Mater. Chem.*, 17:3496–3503, 2007.

[88] Hans Hagemann, Moïse Longhini, Jakub W. Kaminski, Tomasz A. Wesolowski, Radovan Černyý, Nicolas Penin, Magnus H. Sørby, Bjørn C. Hauback, Godwin Severa, and Craig M. Jensen. LiSc(BH4)4: A Novel Salt of Li+ and Discrete Sc(BH4)4 Complex Anions. *J. Phys. Chem. A*, 112(33):7551–7555, 2008. PMID: 18665574.

[89] B. Hammer, L. B. Hansen, and J. K. Nørskov. Improved adsorption energetics within density-functional theory using revised Perdew-Burke-Ernzerhof functionals. *Phys. Rev. B*, 59:7413–7421, Mar 1999.

[90] D. F. Shanno. Conditioning of Quasi-Newton Methods for Function Minimization. *Mathematics of Computation*, 24(111):647–656, July 1970.

[91] J. Voss, J. S. Hummelshøj, Łodziana Z., and T. Vegge. Structural stability and decomposition of Mg(BH4)2 isomorphs an ab initio free energy study. *Journal of Physics: Condensed Matter*, 21(1):012203, 2009.

[92] H. Nöth and P. Fritz. Neue Dibor-Verbindungen. *Angew. Chem.*, 73(11):408–408, 1961.

[93] H.-W. Li, S. Orimo, Y. Nakamori, K. Miwa, N. Ohba, S. Towata, and A. Züttel. Materials designing of metal borohydrides: Viewpoints from thermodynamical stabilities. *J. Alloys Compd.*, 446-447(0):315 – 318, 2007.

[94] ”Hydrogen Storage” presentation by S. Satyapal, 2008 DOE Hydrogen Program, Merit Review and Peer Evaluation Meeting, 9 June 2008. `http://www.hydrogen.energy.gov/pdfs/review08/st_0_satyapal.pdf`, 2012-01-31.

[95] Heinrich Nöth, Egon Wiberg, and Ludwig P. Winter. Boranate und Boranato-metallate. III. Boranatozinkate der Alkalimetalle. *Z. Anorg. Allg. Chem.*, 386(1):73–86, 1971.

[96] Niflheim. https://wiki.fysik.dtu.dk/niflheim/niflheim, 2012-01-31.

[97] Subversion. http://subversion.tigris.org, 2012-01-31.

[98] Cron. http://en.wikipedia.org/wiki/Cron, 2012-01-31.

[99] matplotlib. http://matplotlib.sourceforge.net, 2012-01-31.

[100] Java. ”http://java.com/, 2012-01-31”,.

[101] PyMySQL. https://github.com/petehunt/PyMySQL/, 2012-01-31.