LOBSTER

User’s Guide

Version 5.1.1

The Dronskowski Group
at RWTH Aachen University,
Aachen, Germany,

proudly presents

LOBSTER

Local Orbital Basis Suite Towards Electronic-Structure Reconstruction

A Program to Analyze Chemical Bonding
Based on Plane-Wave and PAW Output

Get your copy at http://www.cohp.de!

The Executive Summary

Thank you for downloading LOBSTER, a program for chemical-bonding analysis! LOBSTER is built to read and process output data from plane-wave DFT packages, such as VASP, ABINIT, Quantum ESPRESSO, and **any other code** supplying its electronic-structure data in a generic format. By re-extracting atom-resolved information from the delocalized plane-wave basis sets, LOBSTER gives you access to projected COOP, COHP, and COBI curves, which you can use to visualize bonding and antibonding contributions in your everyday DFT calculations. Allegedly, LOBSTER can also produce reasonable atom- and orbital-projected densities of states (DOS), in addition to more fancy features such as density-of-energy (DOE), wavefunction-derived charges, polarizations, fatbands, etc. This manual describes how to get going with LOBSTER and explains possible pitfalls. **Please read this manual carefully, especially Section 4.1.**

The newest program version is always available on our website, www.cohp.de. We have also collected some more background information there. In future publications of work performed using LOBSTER, we kindly ask you to properly acknowledge its use by citing the following **references**, depending on the problem:

(1) The original COHP definition: R. Dronskowski, P. E. Blöchl, *J. Phys. Chem.* **1993**, *97*, 8617–8624.

(2) The projected COHP definition: V. L. Deringer, A. L. Tchougréeff, R. Dronskowski, *J. Phys. Chem. A* **2011**, *115*, 5461–5466.

(3) The mathematical apparatus and the framework on which LOBSTER is built: S. Maintz, V. L. Deringer, A. L. Tchougréeff, R. Dronskowski, *J. Comput. Chem.* **2013**, *34*, 2557–2567.

(4) LOBSTER ≥ 2 and its nuts and bolts: S. Maintz, V. L. Deringer, A. L. Tchougréeff, R. Dronskowski, *J. Comput. Chem.* **2016**, *37*, 1030–1035.

(5) LOBSTER ≥ 4: R. Nelson, C. Ertural, J. George, V. Deringer, G. Hautier, R. Dronskowski, *J. Comput. Chem.* **2020**, *41*, 1931–1940.

And now, the unavoidable disclaimer: This software (and manual) is provided AS IS, without the implication that it will be useful, correct, beneficial, or solve all your problems. Use at your own risk!

Table of Contents

1 Getting Started

1.1 Installation

LOBSTER comes in a single binary file. Just copy the `lobster-5.1.1` executable into the directory where your binaries are located (i.e., a directory contained in your `$PATH` variable). We assume that you are using `~/bin` for this:

```
cp lobster-5.1.1 ~/bin/lobster
chmod +x ~/bin/lobster
```

Starting with version 1.1.0, LOBSTER has **no** run-time dependencies anymore and runs (as it is, like in the good old days!) under most contemporary Linux distributions.

1.2 Preparing a VASP calculation

At this time, you should be ready to run your first calculation. Keep in mind that we process plane-wave/PAW output, in this case, generated by VASP. It is **absolutely necessary that this is done in a static run** (no movements of atoms, `NSW = 0`). Remove any previously created `WAVECAR` file and take care to have the new one written to disk (`LWAVE = .TRUE.`).

LOBSTER can now deal with time-reversal symmetry (but not yet other symmetries)—in other words, your VASP `WAVECAR` may contain results for the *entire* mesh or only *half* of it. To get out of trouble, you can either enable time-reversal symmetry or completely switch off symmetry in the `INCAR` (`ISYM = 0 or ISYM = -1,` respectively).

In the current version, **do not** use ultrasoft pseudopotentials (“US-PP”) in your `POTCAR`; please use PAW potentials instead. Also, the gamma-only version of VASP is not supported; please run the single-point calculation with the “default” (complex) version, no matter which **k**-point(s) you are looking at. Finally, we urge you to specify your local basis functions manually as described below, especially when using `_sv` potentials in the single-point calculation. Otherwise spilling will likely be large (see below) **which might render your results questionable!**

1.3 Preparing an ABINIT calculation

In contrast to VASP, ABINIT does not offer a uniform naming scheme of its input and output files. Hence, any post-processing code like LOBSTER is better off with a simple file structure convention: in other words, put only a single ABINIT calculation into one directory. Then, choose an arbitrary name for your calculation, e.g., "diamond", and create a file called `dia-mond.files` to tell ABINIT about its input and output files when started via shell redirection (`abinit < diamond.files`). The naming schemes within that file is totally up to you but it must carry the `.files` suffix.

Otherwise, everything said for VASP also holds for ABINIT as your PAW code of choice. Please **do not** modify the following (default) parameters: To make sure your wavefunctions belong to a static run, prevent both ionic movement (`ionmov 0`) and changes in cell shape and dimensions (`optcell 0`). Also ensure that the wavefunction file is written to disk (`prtwfk 1`).

LOBSTER can now deal with time-reversal symmetry (but not yet other symmetries). That means your wavefunction file may contain results for the *entire* **k**-mesh or only *half* of it – in other words, only `kptopt 3` or `kptopt 0`, respectively, is allowed in your ABINIT input file. Furthermore, LOBSTER does not allow simplifications for real-valued wavefunctions at Γ, and thus please use `istwfk *1` in your ABINIT input file.

Concerning the PAW potentials, LOBSTER only supports PAW datasets provided in the PAW-XML specification. We recommend using the JTH PAW atomic datasets. This requires at least ABINIT version 7.6 but we encourage you to use the latest version anyway.

1.4 Preparing a Quantum ESPRESSO calculation

Just like ABINIT, Quantum ESPRESSO (QE) does not have standardized names for its input and output files. That being said, it is strongly recommended to put a single QE calculation into one directory, then you can choose an arbitrary name for your working directory, e.g., diamond, and create a main input file in it called `diamond.scf.in` to tell QE about its input and output files when started via shell redirection (`pw.x < diamond.scf.in`). The main input file name is arbitrary for QE calculations, but for the purpose of LOBSTER calculations, it has to carry the `.scf.in` suffix. Otherwise, everything said for VASP and ABINIT also holds for QE as your PAW code of choice; its first chemical-bonding application is documented in R. Nelson, P. M. Konze, R. Dronskowski, *J. Phys. Chem. A* **2017**, *121*, 7778–7786.

Furthermore, by default LOBSTER needs the following parameter to be set up inside the `.scf.in` file: `wf_collect = .true.` in the namelist `&CONTROL` to store all wavefunctions into the output data directory, i.e. `"outdir"/"prefix".save`. Additionally, it is **absolutely necessary that the calculation is done in a static run**. In other words, make sure `calculation = 'scf'` or `'nscf'` in the namelist `&CONTROL`.

LOBSTER can now deal with time-reversal symmetry (but not yet other symmetries). That means your wavefunction file may contain results for the *entire* **k**-mesh or only *half* of it – in other words, in the namelist `&SYSTEM` users have to set up `nosym = .true.`, but may set up `noinv = .true.` to switch off symmetry completely or `noinv = .false.` to enable time-reversal symmetry. Furthermore, it is not allowed to use simplifications for, e.g., real-valued wavefunctions at Γ. Therefore, avoid the use of `K_POINTS gamma` for the **k**-point setup. Concerning the PAW potentials, LOBSTER only supports PAW datasets provided in the Unified Pseudopotential Format (UPF).

1.5 Generic Electronic-Structure Inputs (i.e., code whatsoever)

Starting with version 5.0.0, LOBSTER is able to read electronic structures in a **generic** format, allowing the skillful user to write own interfaces to arbitrary electronic-structure codes. All that needs to be done is to parse the necessary info from the electronic-structure output into the generic input format. An example of this generic input is shipped with every copy of LOBSTER. When writing your own interfaces, please also consider making them open to the public, it will surely help the community. Let us all work together!

1.6 A Note on k-Point Symmetry

LOBSTER requires an electronic structure that is described with either an *entire* **k**-mesh or *half* of it (i.e., a mesh using time-reversal symmetry). This, however, does not require the user to do the full self-consistent calculation without the use of symmetry. In a similar spirit to band-structure plot calculations, a self-consistent calculation using full symmetry can performed first, before running a non-self-consistent electronic-structure calculation using different settings for the **k**-mesh based on the charge density of the self-consistent calculation. This approach usually performs much better, as the fully symmetrized **k**-mesh calculations run much faster and numerically more stable than those only using time-reversal symmetry.

1.7 Number of bands

This is an important point, so we have given it a separate section. For pCOHP analyses you need to use at least as many bands as there are orbitals in your local basis, for simple mathematical reasons. By default, PAW codes use fewer bands, and this is fine for all that these codes do; for pCOHP analyses using LOBSTER, however, you need to manually set `NBANDS` in the `INCAR` file of the final run if you use VASP. The corresponding ABINIT and QE parameters are `nband` and `nbnd`, respectively. You find more details on that matter in our 2013 *J. Comput. Chem.* Paper (see references at the beginning of this document). **If you use too few bands, an attempted pCOHP calculation would give nonsense, and LOBSTER refuses to even try.** Always check the number of bands whenever you don't get your desired pCOHP results and take the warnings written out by LOBSTER seriously. The projection, pDOS and pCOOP calculations are not affected and will always be carried out in a LOBSTER run.

1.8 Specifying the local basis

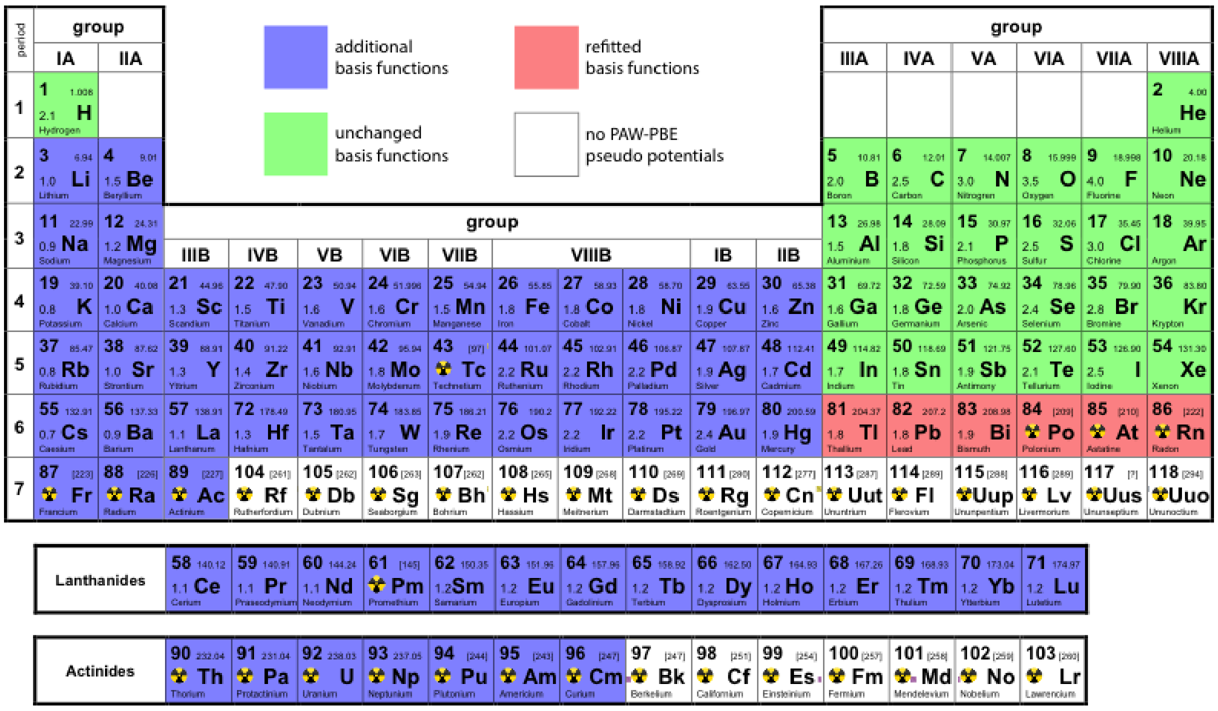
To properly extract the chemistry out of physics, correctly projecting onto individual atoms is mandatory. Hence, employing **a suitable set of basis functions is crucial for every analysis** carried out by LOBSTER! Even though the program is able to make a good guess for the required valence configurations, there are many cases where this guess is not ideal. As a user, it is your responsibility to check your results for plausibility: before starting an analysis, do think about what basis functions *you* would use for every element in your system and compare your expectations to the suggestion printed by LOBSTER at the very beginning of its output. If there are deviations or the yielded spilling comes out high, feel free to experiment a little with the basis sets. The lower the absolute spilling, the more accurate your results, at least in principle. Please note that it is important to get the best (i.e., the smallest) possible absolute **charge** spilling before proceeding with any further bonding analysis. As regards spilling in general, please also look at LOBSTER's list of FAQ which comes with this manual, too.

A good place to start with your considerations about basis functions is given by the electronic configuration of the pseudopotentials, which might very well differ from the valence configuration of a free atom. In case of doubt, please check the electron configuration as given in the PAW dataset you used (e.g., `POTCAR` or the `.xml`-files or the `.UPF`-files).

Sometimes, first-time users experience problems in determining basis functions to be used and find that analyzing the PAW dataset (i.e., pseudopotential files) can be quite complicated. Therefore, starting from LOBSTER 3.1, a new feature has been introduced to supply users with recommendations based on the PAW valence configurations. If such information is unavailable, LOBSTER will recommend these from the atomic valence configurations of the corresponding elements (i.e., the Periodic Table). If users want to directly use the LOBSTER-recommended basis functions, they can do so by putting the keyword `userecommendedbasisfunctions` in their `lobsterin` (see also section 2.3.1). Please note that the feature only provides the minimum set of basis functions. In some calculations users may need to include additional basis functions of the unoccupied orbitals. Please use this feature cautiously and always check your charge spilling.

There are cases where the basis sets initially implanted into LOBSTER turned out as disappointing. For example, take beryllium, whose 2p orbitals are unoccupied in the free Be atom, but are very well involved in the bonding of the bulk metal. To cure that deficiency, LOBSTER 1.2.0 already came with an improved basis set which was augmented for certain elements to improve the projection. This was achieved by adding polarization functions (e.g., 2p for Li) or by fitting certain basis functions to VASP free-atom wavefunctions in a supercell within the GGA-PBE framework (see our 2016 *J. Comput. Chem.* paper). See section 2.3 for details on how to select this basis set.

The following Periodic Table of the Elements shows for which elements basis-set changes have been introduced since LOBSTER 1.2.0:


additional basis functions
refitted basis functions
unchanged basis functions
no PAW-PBE pseudo potentials
group
period
Lanthanides
Actinides

1.9 Running LOBSTER

As soon as your PAW calculation has finished and the data to be processed are written, type

```
~/bin/lobster
```

That's it—watch it run! For a quick start, we provided you with the VASP input files used for our [2013 *J. Comput. Chem.* paper] and an additional spin-polarized case (as well as a few ABINIT and Quantum ESPRESSO input files). These files are in the `examples` folder of the distribution package. Please note that we got our results with a historically grown Fortran version[1] of LOBSTER and have re-written the program from scratch afterwards. It is still founded on the same theories, but uses a more modern technical apparatus. Thus, in some cases numerical deviations may occur, which should of course not affect chemical interpretation. **Before reporting any errors in LOBSTER's results, please do check these tutorial files** to rule out any technical problems on the particular computer you are using. Please do contact us if any of these tutorial files are not working or behave unexpectedly.

1.10 Parallelization

This version of LOBSTER is parallelized using OpenMP (not to be confused with openMPI). That implies that you may harvest the processing power of more than one CPU core as long as

[1] Not a very smart idea but nevertheless requested by Richard Dronskowski at the very beginning (RD).

these cores share the same main memory (RAM). This is always true for a single workstation but also within one compute *node* of a supercomputer. Let's assume your computing center provides you with a bazillion of nodes with 24 CPU cores each: then you can use up to 24 cores per LOBSTER calculation, but not more. You can of course run multiple independent LOBSTER calculations on different nodes.

Per default, LOBSTER will use as many CPU cores as it finds in the computer. In other words, you will *automatically* profit from the parallelization. If you want to force LOBSTER to use a specific number of CPU cores (for whatever good reason), simply set the environment variable `OMP_NUM_THREADS` to the requested number of cores. In C shells this is achieved by executing

```
setenv OMP_NUM_THREADS 24
```

(assuming 24 cores). For bash shells, the respective syntax is:

```
export OMP_NUM_THREADS=24
```

Remember that all this does not change the way LOBSTER has to be handled; it just makes it faster by distributing the work.

1.11 Preparing the k-Path for a Fatband Plot

The path through the Brillouin zone needed to display a fatband plot must be prepared manually. In VASP, the **k**-points forming the path are put together with other **k**-points within the `KPOINTS` file. Hence, instead of using the automatic k-mesh generation, all **k**-points (for an SCF calculation and for the fatband purpose) are entered explicitly (see VASP manual how to do this). To avoid double counting, add the **k**-points for the fatband purpose with a zero weight at the end of the `KPOINTS` file.

In ABINIT, the preparation of the **k**-points for a fatband plot is very similar to the one in VASP, namely by manually entering the fatband **k**-points together with the other **k**-points for an SCF calculation in the input file (see ABINIT manual how to do this). Just like in VASP, you also need to specify the fatband **k**-points with a zero weight.

Just like in the preceding DFT programs, Quantum ESPRESSO requires that all **k**-points are entered manually for a fatband plot in the `K_POINTS` *card* (see pw.x documentation how to do this). Eventually, the fatband **k**-points are added with a zero weight to avoid double counting.

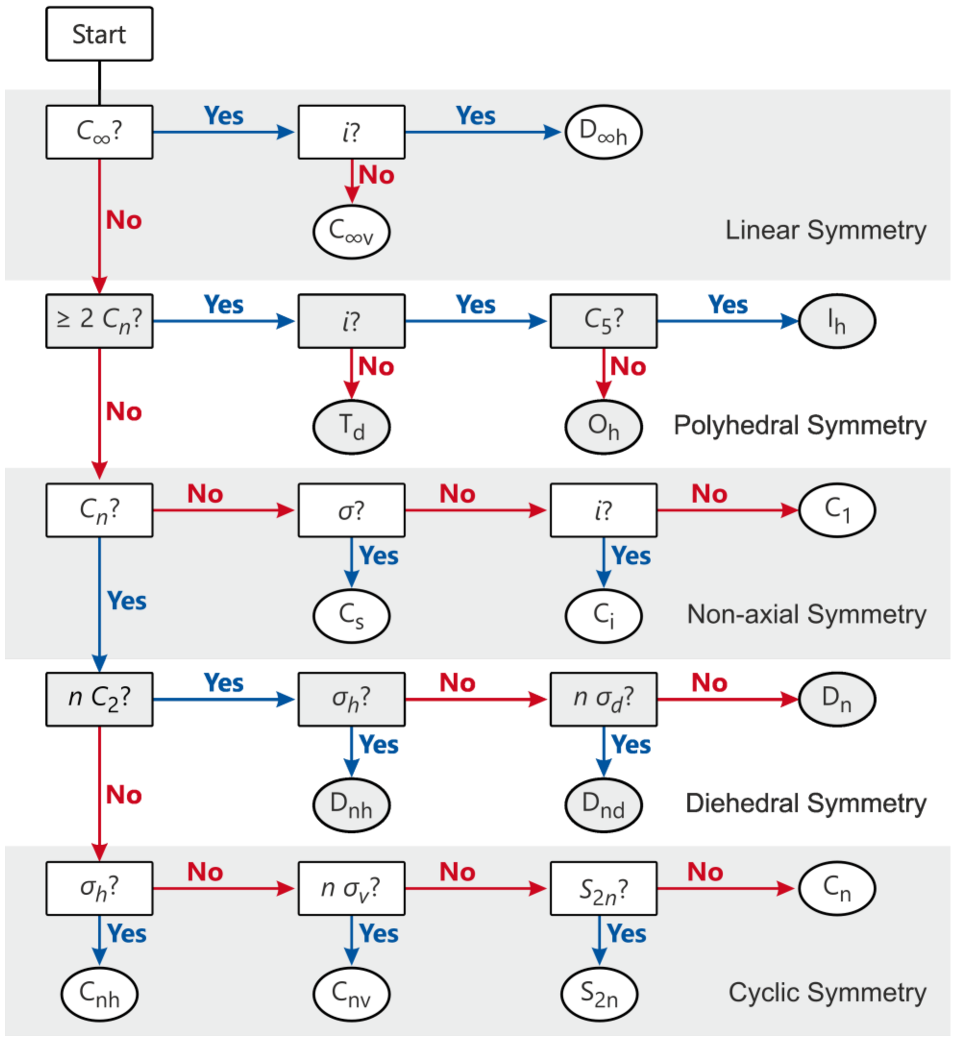
1.12 Molecular and Fragment orbitals

Since version 5.0.0, LOBSTER can calculate localized molecular orbitals derived through embedding. Since version 5.1.0, LOBSTER can also *directly* transform the projected atomic-orbital basis into a canonical *molecular*- or *fragment*-orbital basis with manually defined (by the user) molecules or fragments, in addition to localized molecular orbitals. The latter technique and its new basis allows the use of analytic tools that have been available for the atomic-orbital basis: this so-called LCFO (linear combination of fragment orbitals, see P. C. Müller, N. Schmit, L. Sann, S. Steinberg, R. Dronskowski, *Inorg. Chem.* **2024**, *##*, ####) approach and its basis support projected DOS, COHP, COBI, Löwdin population and charge analyses, fatband plots as well as real-space representations of molecular orbitals and molecular-orbital diagrams. The molecular orbital formation energy (MOFE) is a variant of COHP measuring the band-energy contribution of a given atomic orbital to each individual molecular orbital.

In order to improve usability, an algorithm was implemented that *automatically* determines the point group and character table of the user-defined molecules and matches the molecular orbitals' characters to the irreducible representations in the character table. This way, the canonical molecular orbitals are conveniently labelled according to their symmetry properties. Both character table as well as the orbital characters are printed out for reference. Note that COOP, Mulliken population, Madelung energy and polarization are not accessible due to principle reasons. The definition of the LCFO basis requires minimal user input and works as follows:

1. Manual definition of fragments or molecules by the user
2. Automatic search for symmetry elements in each fragment
3. Assignment of point groups according to a decision tree (see below)
4. Generation of character tables based on the point group
5. Transformation of LCAO to LCFO basis
6. Assignment of Mulliken symbols to each fragment orbital based on the similarity of the fragment orbital's character and the irreducible representations of the character table
7. Printing of fragment orbitals as cube files
8. Bonding analysis analogous to the LCAO basis set.

The following flow chart illustrates the symmetry search for whatever point group:



Start
C_∞?
Yes
i?
Yes
$D_{\infty h}$
No
$C_{\infty v}$
No
Linear Symmetry
$\geq 2\ C_n$?
Yes
i?
Yes
C_5?
Yes
I_h
No
T_d
No
O_h
No
Polyhedral Symmetry
C_n?
No
σ?
No
i?
No
C_1
Yes
C_s
Yes
C_i
Yes
Non-axial Symmetry
$n\ C_2$?
Yes
σ_h?
No
$n\ \sigma_d$?
No
D_n
Yes
D_{nh}
Yes
D_{nd}
No
Diehedral Symmetry
σ_h?
No
$n\ \sigma_v$?
No
S_{2n}?
No
C_n
Yes
C_{nh}
Yes
C_{nv}
Yes
S_{2n}
Cyclic Symmetry

2 The `lobsterin` File

2.1 How the input works

LOBSTER is controlled by a single file, called `lobsterin`. This file is designed such that it needs only minimal user input. Usually, the default parameters are fine: you should only need to set the energetic window, the local basis (e.g., “4s 4p” for gallium, or also “4s 4p 3d”, depending on your plans). The `lobsterin` file also contains those atom pairs for which you intend to do pCOOP/pCOHP/COBI analysis. If no atom pairs at all are specified, LOBSTER will nonetheless and automatically try to find any pairs between 1.0 and 3.0 A distance. However, chemistry is about chemical interactions between atoms, so you need to manually specify pairs of atoms to arrive at pCOOPs or pCOHPs or COBIs which you intend to analyze.

The `lobsterin` file is `not cAsE sEnSiTiVe`, and comments may be added using an exclamation mark (`!`), a hash (`#`) or a double slash (`//`). Everything in a line *before* a comment symbol, however, will be read and processed!

2.2 An example input file

```
! This is an example for the lobster control file lobsterin.
! (See, here we are using the comment function!)
!
! First, enter the energetic window in eV (relative to the Fermi level):
COHPStartEnergy -10
COHPEndEnergy 5
!
! Then, specify which types of valence orbitals to use:
includeOrbitals s p d
! You can also specify the basis functions per element manually, e.g.:
! basisFunctions Ga 4s 4p
! basisFunctions Sr 5s 4s 4p ! Sr_sv potential
!
! Now define the pairs for which COHP analysis etc. should be done.
! The atoms are numbered as per order in the PAW-code control file.
cohpBetween atom 1 atom 10
!
! If you are interested in single orbital COHPs, you can get all the pairs
! like s-s, s-p_x, ..., p_z-p_z. Uncomment this line to switch it on:
! cohpBetween atom 1 atom 2 orbitalWise
!
! If you are interested in a bond that involves an atom that is not located
! in the original but in a neighboring unit cell, you can shift it using the
! cell keyword
!
! cohpBetween atom 1 atom 2 cell 1 0 0
!
```

```
!
! If you want to generate the COHP pairs automatically, use this to include
! all pairs in a given distance range (in Angstrom, not in atomic units):
! cohpGenerator from 1.4 to 1.5
! cohpGenerator from 1.4 to 1.5 type Ga type Sr
! and in the latter case only between the specified elements
!
! If you want to transfer the LCAO basis set into a molecular- or fragment-
! orbital basis set, you can tell LOBSTER to combine any number of atoms:
! molecule atom 1 atom 2 atom 3 cell 1 0 0
!
! If molecules or fragments are defined, COHP pairs can be generated as
! follows:
! cohpGenerator from 1.4 to 1.5 type Ca type CO3
! Note that the type definition of molecules is case sensitive!
! E.g., CO will be interpreted as carbon monoxide, Co as cobalt.
! The order of atoms is arbitrary: E.g., CO3 = COOO = O3C = O2CO; and so on
```

2.3 List of keywords

2.3.1 Standard commands

All of these case-insensitive parameters are optional, but useful at times. New keywords in LOBSTER 5.1.0 are highlighted in **bold**.

`basisSet`	If you wish to override the basis set selected by default, you may do so with this keyword. It currently supports `bunge` (for elements up to Xe), `koga` and `pbeVaspFit2015` (up to Lr; yes, that is $Z = 103$). The references are: • [`bunge`] C. Bunge, J. Barrientos, A. Bunge, *At. Data Nucl. Data Tables* **1993**, *53*, 113–162. • [`koga`] a) T. Koga, K. Kanayama, S. Watanabe, A. J. Thakkar, *Int. J. Quant.* Chem. 1999, 71, 491–497, b) T. Koga, K. Kanayama, T. Watanabe, T. Imai, A. J. Thakkar, Theor. *Chem. Acc.* **2000**, *104*, 411–413. • [`pbeVaspFit2015`] based on [`koga`] but with additional functions fitted by S. Maintz to atomic VASP GGA-PBE wavefunctions; S. Maintz, V. L. Deringer, A. L. Tchougréeff, R. Dronskowski, *J. Comput. Chem.* **2016**, *37*, 1030–1035. • [custom] a custom basis defined in a separate file. See keyword “`CustomSTOforAtom`”.

`basisFunctions`	This repeatable keyword lets you specify the employed basis functions for each element; see `lobsterin` file for syntax.
`cohpBetween`	This repeatable keyword selects a pair of two atoms for bonding analysis; see `lobsterin` file for syntax. If the **`molecule`** keyword is defined, in the `lobsterin` file, bonds involving molecules can be defined using: `cohpbetween frag 1 frag 2` or `cohpbetween frag 1 atom 2`; in case atom 2 is not part of a molecule
`cohpGenerator`	This keyword makes LOBSTER build the list of atom pairs to be analyzed using given distance or element type criteria. See `lobsterin` file for syntax.
`orbitalWise`	Appending this keyword after the `cohpBetween` or `cohpGenerator` command will make LOBSTER calculate the orbital-resolved (instead of atom-resolved) pCOOP and pCOHP. This is also compatible with the LCFO basis set.
`around`	Appending this keyword followed by an atom index after the `cohpGenerator` command will make LOBSTER calculate the pCOOP and pCOHP only for interactions formed by the atom specified by that given index. If molecules are defined, this keyword adds the interaction formed by the respective molecule.
`COHPStartEnergy`	Energy in eV where bonding analysis (pDOS, pCOOP and pCOHP) starts, relative to the Fermi level.
`COHPEndEnergy`	Energy in eV where bonding analysis (pDOS, pCOOP and pCOHP) stops, relative to the Fermi level.
`COHPSteps`	The number of energy increments for your bonding analysis; this is especially useful if your energy window is much larger or narrower than average.
`forceEnergyRange`	LOBSTER automatically adjusts the energy boundaries such that the Fermi level (exactly zero eV) is included. This option disables that automatic adjustment, but may disable the output of the `ICOHPLIST.lobster` file.

`gaussianSmearingWidth`	If using Gaussian broadening for energy integration (equivalent to `ISMEAR = 0` in VASP), you may here select the smearing width in eV. Do not hesitate to play with this a little! The default is `0.2` (eV), and LOBSTER does *not* read your PAW code setting. Keep in mind that you may have to increase the number of **k**-points in your PAW calculation to use very low smearing.
`DensityOfEnergy`	This generates the density-of-energy (DOE) function, to be understood as a generalized COHP; the DOE comprises *all* elements of the density-of-states matrix $P_{\mu\nu}$ and the Hamiltonian matrix $H_{\mu\nu}$, summing up all off-site (inter-atomic) and on-site (atomic) contributions. Hence, we energetically resolve the entire band-structure energy which results as the DOE's energy integral. The DOE has been introduced in M. Küpers, P. M. Konze, S. Maintz, S. Steinberg, A. M. Mio, O. Cojocaru-Mirédin, M. Zhu, M. Müller, M. Luysberg, J. Mayer, M. Wuttig, R. Dronskowski, *Angew. Chem. Int. Ed.* **2017**, *56*, 10204–10208. The output is written to `DensityOfEnergy.lobster`.
`BWDF`	This enables evaluation of the bond-weighted distribution function (BWDF), which may be useful when looking at very complex or even amorphous structures (see reference below); at the moment, the BWDF is weighted using integrated pCOOPs. The keyword can be followed by the binning interval that defaults to 0.02. The BWDF has been introduced in V. L. Deringer, W. Zhang, M. Lumeij, S. Maintz, M. Wuttig, R. Mazzarello, R. Dronskowski, *Angew. Chem. Int. Ed.* **2014**, *53*, 10817–10820. The output is written to `BWDF.lobster`.
`BWDFCOHP`	This does the same as the `BWDF` keyword (see above) but the weights are the integrated pCOHPs. The output is written to `BWDFCOHP.lobster`. Please note that the `BWDF` keyword *also*

	must be specified in lobsterin in order to activate the `BWDFCOHP` key.
`skipDOS`	If, for some reason, you do not want the density-of-states, this keyword will make LOBSTER skip the calculation and not write the density-of-states to disk.
`skipCOOP`	Similar to `skipDOS`, this keyword will make LOBSTER skip the calculation and not write pCOOP and IpCOOP to disk.
`skipCOHP`	Similar to `skipDOS`, this keyword will make LOBSTER skip the calculation and not write pCOHP and IpCOHP to disk.
`skipCOBI`	Similar to `skipDOS`, this keyword will make LOBSTER skip the calculation and not write COBI and ICOBI to disk.
`skipPopulationAnalysis`	Since the Mulliken and Löwdin population analysis is done by default and the results are written into the file `CHARGE.lobster`, it can be skipped with this keyword.
`skipGrossPopulation`	The orbitalwise Mulliken and Löwdin gross population for every atom are written into a file named `GROSSPOP.lobster` by default. You can skip it with this keyword.
`skipMadelungEnergy`	The electrostatic energies and site-potentials are written into the `MadelungEnergies.lobster` and `SitePoten-tials.lobster` files by default. Setting up this keyword skips their calculation and output.
`skipCar`	If, for some reason, you are not interested in the energy-resolved COOP, COHP and COBI, you can skip this and print only the files containing the energy integrals. This method will be more time-efficient as only the value at the Fermi energy will be calculated.
`useRecommendedBasisFunc-tions`	This keyword will directly employ the basis functions recommended by LOBSTER. Note that this keyword takes precedence over the keywords `basisFunctions` and `in-cludeOrbitals`.

`skipProjection`

If, for some reason, you just want to deal with the PAW part this keyword will make LOBSTER stop the calculation before the projection. This can be useful if you, for instance, only want to plot PAW wavefunctions or look at the recommended basis functions, etc.

`printLmosOnAtoms`

Followed by indices of targeted atoms, this keyword will calculate embedded localized molecular orbitals as formulated in M. Pauls, D. Schnieders, R. Dronskowski, *J. Phys. Chem. A* **2023**, *127*, 6541–6551 and print those centered on the requested atoms to file in a cube file format.

`molecule`

This keyword is needed to invoke molecular- or fragment-orbital analysis as defined in P. C. Müller, N. Schmit, L. Sann, S. Steinberg, R. Dronskowski, *Inorg. Chem.* **2024**, *##*, #### since it specifies the list of atoms the user decides to “constitute” a molecule or fragment. `molecule` accepts any number of atoms; the only limitation is that each atom can only be defined once in the molecular-orbital basis set, for logical reasons. Please cite the aforementioned paper whenever the feature is used.

If atoms are not located in the original unit cell, the corresponding translation vector can be added by the keyword “`cell`” like this:

```
molecule atom 1 atom 2 atom 3 cell 1 0 0
```

If this keyword is set, a character table is automatically generated and printed, in addition to the molecular orbitals in a cube file format. If interactions were defined with either the `cohpBetween` or `cohpGenerator` keywords, a bonding analysis is performed.

`skipMOFE`

This skips the calculation of the molecular orbital formation energy (MOFE).

`skipMolecularOrbitals`

This skips generation of the cube files containing the molecular orbitals generated by the LCFO basis set.

2.3.2 For experts only

Please, only use these keywords if you know what you are doing. **You have been warned!**

`basisRotation`	This enables rotation of the basis functions, useful for orbital-resolved bonding analysis (S. Maintz, M. Esser, R. Dronskowski, *Acta Phys. Pol. B* **2016**, *47*, 1165–1174). The keyword must be followed by 3 numbers defining the vector of the rotation axis and the angle in degrees.
`autoRotate`	Engages a novel algorithm that spatially aligns all basis functions to the coordination polyhedra of each atom individually. The rotation parameters are determined fully automatically and then send to a modified version of the mathematical scheme from the reference given for `basisRotation`.
`createFatband`	The keyword will make LOBSTER calculate **k**-dependent energy values of the various bands alongside a descriptor measuring the contribution of the element and orbital(s) chosen. The keyword must be followed by the element symbol and orbitals' label in question. There are two ways to specify orbitals' label. First, one can specify the principal quantum number (*n*) and only the letter label of the orbital quantum number (*l*) just like specifying elements and orbitals for the `basisFunctions` keyword. The other way is by specifying the principal quantum number together with the letter labels of the orbital and magnetic quantum numbers (*l* & *m*) which in LOBSTER are set as the following: `s`, `p_x`, `p_y`, `p_z`, `d_xy`, `d_xz`, `d_yz`, `d_z^2`, `d_x^2-y^2`, `f_xyz`, `f_xz^2`, `f_yz^2`, `f_z^2`, `f_x(x^2-3y^2)`, `f_y(3x^2-y^2)`, `f_z(x^2-y^2)`. An example input would be: `createFatband Si 3s 3p` or `createFatband Si 3s 3p_x 3p_z`.

Our method is demonstrated in M. Esser, V. L. Deringer, M. Wuttig and R. Dronskowski, *Solid State Commun.* **2015**, *203*, 31–34.

The output(s) is written to `FATBAND_AX_B.lobster`. Here `A` and `X`, respectively, denote the atom symbol and index, whereas `B` denotes the orbital label. See Section 1.9 for all details on how to prepare the path through the Brillouin zone needed for displaying a fatband plot and Section 3.1 on the output file format; you may also glimpse into the appendix.

`writeBasisFunctions`

If, for some reason, you want to take a look at the basis functions LOBSTER employs, this keyword writes their parameters to a file called `basisFunctions.lobster`.

`writeMatricesToFile`

This keyword makes LOBSTER write the overlap, Hamiltonian, transfer and coefficient matrices for each spin and **k**-point to files. Note that there will be three variants of the coefficient matrices: the original ones and one for each of the applied orthonormalization steps (see our 2016 *J. Comput. Chem.* paper).

`saveProjectionToFile`
`loadProjectionFromFile`

These two keywords allow you to save the results of the projection or load them to do differently set up bonding analysis without recalculating the projection within an unchanged basis. Do *not* use this option if you want to change the basis, though. The data is written to a binary file called `projectionData.lobster`.

`printTotalSpilling`

This keyword will instruct LOBSTER to print the absolute total spilling next to the absolute charge spilling.

`realSpaceHamiltonian`
`realSpaceOverlap`

These keywords make LOBSTER calculate the Fourier-Transforms of either matrix and yield the real-space representations of H(**T**) and S(**T**), where **T** are translational vectors. If you append “`layers n`” to the keyword, the matrices will be calculated for all neighboring unit cells up to *n* layers.

`printPAWRealSpaceWavefunction kpoint`

This keyword makes LOBSTER calculate the real-space representation of the PAW wavefunction of a **k**-point specified by an integer following the keyword and write its values to a file. The first three columns of this file carry the Cartesian coordinates within the unit cell. Column four gives the distance from the current point to the first point, column 5 and 6 give the real and imaginary values of the wavefunction, respectively.

You can specify the starting and end points of a line by appending either one of the following means:

- `coordinates X1 Y1 Z1 coordinates X2 Y2 Z2`
- `atom A1 atom A2`

`X`, `Y` and `Z` are given in fractional coordinates within the unit cell and `A` refers to the index of an atom as ordered in the PAW calculation. If you use the `atom` keyword, you can put `stretchLine` after the second index to stretch the line to its longest possible extent through the unit cell.

If you append `box`, LOBSTER will treat the specified coordinates as two points spanning a cuboid within the unit cell yielding a 3D grid in the output file. You have to make sure the cuboid is completely covered by the unit cell!

Appending `pointsPerAngstrom P` changes the resolution of the visualization to the integer value specified in `P`. Default is 100 for line plots and 10 for box plots.

Ending the line with `bandList` followed by a list of integers makes LOBSTER evaluate the visualization for the specified set of bands only, instead of treating all that are available.

`printLCAORealSpaceWavefunction kpoint`	This command is analog to `printPAWRealSpace-Wavefunction kpoint`, but gives the projected LCAO wavefunctions. It produces two sets of files, one before the orthonormalization step and one after.
`noFFTForVisualization`	By default, the PAW wavefunction visualization uses a fast Fourier-Transform algorithm that is far more efficient than its discrete, potentially more accurate analog. This keyword enforces the latter algorithm.
`RMSp`	Enables the calculation of the root-mean-square of the projection (RMSp). Please see our 2016 *J. Comput. Chem.* paper for its definition. Results are written to `RMSp.lobster`. Note that calculating the RMSp slows down the projection.
`onlyReadVasprun.xml`	When operating on VASP input data, this keyword forces LOBSTER to read all values only from `vasprun.xml`, even if other files offer numerically more accurate values.
`noMemoryMappedFiles`	The binary file of the PAW wavefunctions can be *cached* by LOBSTER very efficiently to improve performance. If, for any reason, this uses too much memory, you can disable this behavior. Please remember, however, that a cache is freed automatically if the memory is used otherwise.
`skipPAWOrthonormalityTest`	By default, LOBSTER checks the bands at each **k**-point of your input PAW wavefunctions for orthonormality. This takes a very small amount of time and may be suppressed but we strongly recommend not to do it, unless you repeat various calculations using the same input data!
`doNotIgnoreExcessiveBands`	If the PAW wavefunctions contain more bands than there are basis functions in the local basis, those excessive bands are ignored by default. This keyword

	forces LOBSTER to project them as well, but in turn disables pCOHP analysis.
`doNotUseAbsoluteSpilling`	Starting with version 2.0.0, LOBSTER prints the absolute charge spilling (occupied levels) and the absolute total spilling (all levels, even unoccupied ones), which are defined similar to the original spilling, but averages the absolute values over bands and **k**-points. In certain cases, the original method which is switched on by this keyword can result in unphysical (negative) values and should hence be avoided.
`skipReOrthonormalization`	By default, LOBSTER re-orthonormalizes the projected wavefunctions (see our *J. Comput. Chem.* 2013 paper). We recommend **against** disabling this unless you know exactly about the consequences!
`doNotOrthogonalizeBasis`	To yield pCOHP curves that best compare to traditional (LMTO) results, applying Löwdin's symmetric orthogonalization (LSO) to the basis functions is crucial. This is the *default* since LOBSTER 2.0.0, for good reasons. Please also note that the projected bands are orthogonalized as well (see `skipReOrthonormalization` keyword). To disable LSO with respect to the basis functions, use this command (*not* recommended except for specialist use).
`forceV1HMatrix`	In combination with `doNotOrthogonalizeBasis` and `noSymmetryCorrection` this gives pCOHPs like in LOBSTER v1.x.y. Use this only for consistency checks!
`useOriginalTetrahedronMethod`	When employing the tetrahedron method, LOBSTER automatically calculates the pDOS, pCOOP and pCOHP values by differentiating their integrals, thereby yielding a better one-to-one correspondence if compared with the original algorithm. This keyword makes LOBSTER nonetheless use the original tetrahedron method.

`useDecimalPlaces`	By default, LOBSTER prints 5 decimal places to the values in its output files. This keyword followed by an integer number allows you to modify that accuracy.
`kSpaceCOHP`	This keyword will make LOBSTER calculate **k**-dependent COHP. The output(s) is written to `KspaceCOHPBandX.lobster`. Here, `X` denotes the band index. Furthermore, LOBSTER only outputs COHP of the same **k**-points used for fatband, i.e., **k**-points with a zero weight. See Section 1.9 for all details on how to prepare a **k**-path for displaying a fatband plot and Section 3.1 on the output file format. This powerful method has been introduced in X. Sun, X. Li, J. Yang, J. Xi, R. Nelson, C. Ertural, R. Dronskowski, W. Liu, G. J. Snyder, D. J. Singh and W. Zhang, *J. Comput. Chem.* **2019**, *40*, 1693–1700.
`LSODOS`	This keyword activates the DOS output from the orthonormalized LCAO basis corresponding to the Löwdin population analysis. As this basis is the same as for pCOHP/COBI calculations, it might be beneficial to compare this DOS to the pCOHP/COBI plots.
`cobiBetween`	This keyword specifies the list of atoms contributing to a given multicenter bond. `cobiBetween` accepts any number of atoms, but please keep in mind that a large number can lead to an enormous computational workload and, also, lots of numerical noise due to error propagation. If atoms are not located in the original unit cell, the corresponding translation vector can be added by the keyword "`cell`" like this: `cobiBetween atom 1 atom 2 atom 3 cell 1 0 0`

	The "`cobiBetween`" keyword cancels the calculation of a COBI average as it has no scientific meaning for interactions containing different numbers of atoms.
`EwaldSum`	This keyword allows for the definition of multiple additional Ewald splitting parameters for calculating electrostatic site-potentials and lattice energies. This is not strictly needed as LOBSTER automatically determines a default value but should be used for consistency checks. Syntax: `EwaldSum 2 4`

`customSTOforAtom`	This keyword allows to define custom STO basis sets for atoms. The syntax is as follows:
	`useBasisSet custom`
	`customSTOforAtom [ELEMENT] [BASIS_FILE]`
	Where `[ELEMENT]` needs to be replaced by the element symbol and `[BASIS_FILE]` is a basis set file residing within the working directory that defines the basis. An example of such basis set file is shipped with every copy of LOBSTER.

`printMofeAtomWise`	In principle, MOFE contains all combinations of atomic and molecular orbitals in the molecule but which may be difficult to interpret. This keyword prints out the atom-based MOFE by summing over all atomic orbitals. Please see P. C. Müller, N. Schmit, L. Sann, S. Steinberg, R. Dronskowski, *Inorg. Chem.* **2024**, *##*, *####* for more details.
`printMofeMoleculeWise`	Same as above, but for a molecule-based MOFE.
`writeAtomicOrbitals`	Within the LCFO basis, only orbitals of fragments with more than one atom are printed out by default. If you are interested in the atomic orbitals as well, you can print them using this keyword.

`gridDensityForPrinting`	LOBSTER uses a default of 0.05 Å for step size in its cube and charge files. If you want to use a different value, you can enter it (in Å) using: `gridDensityForPrinting 0.1`
`gridBufferForPrinting`	If an atom is close to the border of the unit cell, the orbitals printed out by LOBSTER will be cut off. This keyword will allow you to add some vacuum (in Å) to the boundary. If no value is entered, LOBSTER will determine a default that ensures each atom has a minimum distance of 3 Å to the boundary. Use: `gridBufferForPrinting 1`
`writeAtomicDensities`	If you want to have a look at the electron density of the atomic orbitals in your unit cell, you can use this keyword followed by the indices of all atoms you are interested in. The procedure follows Mulliken's population analysis and is described and also illustrated in P. C. Müller, S. R. Elliott, R. Dronskowski, R. O. Jones, *J. Phys.: Condens. Matter* **2024**, *36*, 325706.

3 Output and Visualization

3.1 Output files

If everything went smoothly, LOBSTER provides you with the following files:

- `lobsterout`: This is the general output file, and it duplicates all the information which has also been written to your terminal. It is a good idea to keep this file as a receipt, for looking up spilling values, and so on.
- `POSCAR.lobster`: This file contains the atomic structure in the POSCAR file format.
- `DOSCAR.lobster`: This file contains the orbital-projected electronic DOS and their sum. The format is similar to VASP's `DOSCAR` file *but the energy has been shifted such that the Fermi level lies at zero eV.* In case of doubt, please see the original description of the format (http://cms.mpi.univie.ac.at/vasp/vasp/DOSCAR_file.html).

- **`DOSCAR.LSO.lobster`**: same as above, but calculated from the orthonormalized LCAO basis.
- **`COHPCAR.lobster`**: File that contains the pCOHPs as requested in the `lobsterin` file. It resembles the format of TB-LMTO-ASA's `COPL` file, which is organized as follows:
  - Starting in line 3, the labels for the interactions are presented, followed by the actual data.
  - Column 1: energy axis, *shifted such that the Fermi level lies at zero eV.*
  - Column 2: pCOHP *averaged* over all atom pairs specified
  - Column 3: integrated pCOHP (IpCOHP) *averaged* over all atom pairs
  - Column 4: pCOHP of the first interaction
  - Column 5: IpCOHP of the first interaction
  - and so on...

  Note that in a spin-polarized calculation, the first set of the columns (2, 3, …, $2N$+3) belongs to the first (up) spin and the other set ($2N$+4, $2N$+5, …, $4N$+5) belongs to the second (down) spin. Here N is the number of interactions.
- **`COOPCAR.lobster`**: same as above, just for pCOOP and its integral (IpCOOP).
- **`COBICAR.lobster`**: same as above, just for COBI and its integral (ICOBI). This feature has been introduced in P. C. Müller, C. Ertural, J. Hempelmann, R. Dronskowski, *J. Phys. Chem. C* **2021**, *125*, 7959–7970. Please cite the paper accordingly whenever this feature is used.
- **`ICOHPLIST.lobster`**: gives you a list of the particular IpCOHP values, integrated up to the Fermi level, for each atom–atom interaction you specified. The data are organized as follows:
  - The first line describes the column titles.
  - The second line contains information on the spin channels for each IpCOHP
  - Column 1: interaction index.
  - Column 2: the index of the first atom that creates the interaction in question (always located in the unit cell at the origin, i.e., T =[0,0,0]).
  - Column 3: the index of the second atom that creates the interaction in question (may locate in a different unit cell from the first atom).
  - Column 4: the distance of the first atom from the second one.
  - Column 5: the vector connecting the unit-cell of the first atom with the one of the second atom.

    - Column 6: IpCOHP of the interaction in question calculated up the Fermi level for the first spin channel.
    - Column 7: IpCOHP of the interaction for the second spin channel if a spin-polarized wavefunction is provided.
- **`ICOOPLIST.lobster`**: same as above, just for IpCOOP.
- **`ICOBILIST.lobster`**: same as above, just for ICOBI. As above, this feature is also described in P. C. Müller, C. Ertural, J. Hempelmann, R. Dronskowski, *J. Phys. Chem. C* **2021**, *125*, 7959–7970. Please cite the paper accordingly whenever this feature is used.
- **`DOSCAR.LCFO.lobster`**, **`COHPCAR.LCFO.lobster`**, **`COBICAR.LCFO.lobster`**, **`ICOHPLIST.LCFO.lobster`**, **`ICOBILIST.LCFO.lobster`**, **`GROSSPOP.LCFO.lobster`**, **`CHARGE.LCFO.lobster`**: The same as above, but for the LCFO basis set described in P. C. Müller, N. Schmit, L. Sann, S. Steinberg, R. Dronskowski, *Inorg. Chem.* **2024**, *##*, ####.
- **`BWDF.lobster`**: this optional file lists the distance in column 1 and the according BWDF values in column 2. If available, column 3 gives values for the second spin channel.
- **`FATBAND_AX_B.lobster`**: The output is initiated by a commentary line, stating the projected element and orbital as well as the number of bands. The subsequent content is organized as follows:
    - The first column gives the index of the **k**-points, ordered in unison with the `KPOINTS` file. Each new **k**-point output is preceded by a commentary line giving its fractional coordinates.
    - The second column gives all the energy eigenvalues (in eV) of the various bands at the **k**-point indexed in the first column. The energies are shifted so that the Fermi level equals zero.
    - In the third column, the contribution of the chosen orbital participating in the band is given. The values range from zero (for no contribution at all) to unity (for complete dominance) of the band character in question.
- **`DensityOfEnergy.lobster`**: this is also an optional file containing the density-of-energy, DOE. The format is exactly the same as **`DOSCAR.lobster`** for the total DOS.
- **`CHARGE.lobster`**: this default file lists the charges from Mulliken and Löwdin population analyses as `Mulliken charge` and `Loewdin charge` for every atom and also gives the total charge as a sum over all atomic charges. The Mulliken and Löwdin population analyses based on plane waves have been introduced in C. Ertural, S. Steinberg, R. Dronskowski, *RSC Adv.* **2019**, *9*, 29821–29830.

- `GROSSPOP.lobster`: this default file lists the Mulliken (`Mulliken GP`) and Löwdin gross orbital population (`Loewdin GP`) for every atom and gives the total atomic gross population as a sum over all gross orbital populations, reflecting the total number of (valence) electrons for that given atom. If a spin-polarized wavefunction is used for the projection, the populations will be given for both spins individually.
- `POLARIZATION.lobster`: this file contains the polarization vector and the absolute polarization, based on the Mulliken and Löwdin charges, averaged over different orientations of the cell, as applied in L. Reitz, P. C. Müller, D. Schnieders, R. Dronskowski, W. I. Choi, W.-J. Son, I. Jang, D. S. Kim, *J. Comput. Chem.* **2023**, *44*, 1052–1063. Please cite this paper whenever the feature is used.
- `SitePotentials.lobster`: This file contains the electrostatic site-potential (in V) for every atom as calculated from Mulliken and Löwdin charges. Additionally, the total electrostatic (often called *Madelung*) energy (in eV) is added at the end. The table is repeated for each manually entered Ewald splitting parameter. Please cite P. C. Müller, C. Ertural, J. Hempelmann, R. Dronskowski, *J. Phys. Chem. C* **2021**, *125*, 7959–7970 whenever this feature is used.
- `MadelungEnergies.lobster`: This file contains the total electrostatic (*Madelung*) energies (in eV) for each entered Ewald splitting parameter as calculated from Mulliken and Löwdin charges. Please cite P. C. Müller, C. Ertural, J. Hempelmann, R. Dronskowski, *J. Phys. Chem. C* **2021**, *125*, 7959–7970 whenever this feature is used.
- `KspaceCOHPBandX.lobster`: these optional files list **k**-resolved COHP for considered bands. Here, `X` in the filename denotes the index of the corresponding band. The subsequent content of each file is organized as follows:
  - The first line is a header which describes the number of spins and **k**-points.
  - The next lines are the data which are grouped in blocks.
  - Each block, which describes data of one **k**-point, starts with a header providing information about spin, energy eigenvalue, the **k**-point index and fractional as well as (orthogonal) cartesian coordinates.
  - Starting from the second line of each block, the COHP data are presented in the form of a matrix which is quite natural to describe interactions of orbital pairs.
- `NcICOBILIST.lobster`: This file contains the ICOBI for all two-center as well as multi-center COBI interactions and consists of four columns.
  - Column 1: interaction index.
  - Column 2: number of atoms in the respective interaction.

  - Column 3: ICOBI calculated up to the Fermi level.
  - Column 4: element symbol, index and translation vector for each atom.
- **`LCFO_Fragments.lobster`**: contains fractional coordinates of the atoms corresponding to the fragments used for the LCFO basis set.
- **`AxBy.Symmetry.lobster`**: contains the symmetry information of a manually defined molecule. The first part contains the character table of the molecule, followed by the characters of each molecular orbital for each spin channel. The bottom part lists the symmetry elements that were found by LOBSTER. For (improper) rotations, the rotation axis is given; for reflections, the normal vector of the mirror plane is given.
- **`AxBy.MO_Diagram.lobster`**: contains the information needed to construct an MO diagram for the molecule. The information is given as a matrix with the LCAO coefficients of each molecular orbital. The top row (most left column) contains the eigenvalues of the molecular (atomic) orbitals.
- **`MOFECAR.lobster`**: if the `lobsterin` file contains manually defined molecules using the `molecule` keyword, this file is written out. It contains the molecular orbital formation energy (MOFE) as defined in P. C. Müller, N. Schmit, L. Sann, S. Steinberg, R. Dronskowski, *Inorg. Chem.* **2024**, *##*, #### in the same format as the `COHPCAR.lobster` file.
- **`IMOFELIST.lobster`**: This file contains the integrated molecular orbital formation energies within the LCFO basis set. Its format is analogous to the `AxBy.MO_Diagram.lobster` file.

3.2 Visualizing your results

Now it is time to look at your results. As the output formats are designed to resemble those of VASP, TB-LMTO-ASA etc., you could just employ the same visualization software you already prefer for looking at those `DOSCAR` or `COPL` files. It is possible to use `gnuplot`; this route, however, is currently a little cumbersome unless you are comfortable with tools like `head`, `tail`, `cut`, and `paste`. Hence, we *strongly suggest* you try the visualization program wxDragon, of which a free trial version is available at www.wxdragon.de. In its last version, wxDragon has been greatly improved to support most of LOBSTER's outputs (please see its manual for details). To start the visualization with wxDragon simply open the files by typing

```
wxdragon DOSCAR.lobster
wxdragon COHPCAR.lobster
```

Visualizing the wavefunctions (3D only) is also possible with wxDragon (please see option `box` in `printPAWRealSpaceWavefunction` or `printLCAORealSpaceWavefunction`). To do so with VASP, simply follow these steps:

- type `wxdragon POSCAR` (or `OUTCAR`),
- then `CTRL-m`,
- click the "`Modules`" button in the new dialog.

After reading the various files, the surface-plot dialog opens for visualizing the wavefunctions. The same steps can be followed with ABINIT and Quantum ESPRESSO, the only difference being the first step; instead of typing `wxdragon POSCAR`, type `wxdragon 'case'.in` or `wxdragon 'case'.out` with ABINIT, and type `wxdragon 'case'.scf.in` or `wxdragon out` with Quantum ESPRESSO. In ABINIT, `'case'.out` is the output file listed in the `.files` file whereas in Quantum ESPRESSO the file `out` is the output redirected from the stdout, i.e., `pw.x < 'case'.scf.in > out`.

4 General Questions

4.1 Chemistry vs. physics: how reliable are my results?

LOBSTER reconstructs local quantities (i.e., chemistry) from PAW results (i.e., physics) via a projection technique which is not a black-box scheme but described in detail in the third reference (see page 3). We reiterate that, at the moment, we are using a *minimal* basis. Even though this basis fits quite well for a wide range of systems, there can be no guarantee that this basis set is also suitable for your specific system. There is a rich "basis-set history" in quantum chemistry, and we will continue to offer better basis sets in the future. Hence, LOBSTER 5.0 follows that idea by offering customized basis sets to be used.

Hence, **it is absolutely necessary that you validate your results**. Beginners in chemical-bonding analysis should check (p)COHP curves for a simple system with similar ones from the literature, or TB-LMTO-ASA computations, **only then** should they move on to very large systems. **Use your chemical intuition all the time under all circumstances.** Additionally, LOBSTER provides you with two numerical hints that should make you think twice about your results: a high spilling value and a potentially produced `bandOverlaps.lobster` file.

Let's assume LOBSTER prints "`abs. charge spilling: 5.00%`" in the `lobsterout` file. This means that the projected electronic structure deviates from the original wave function by 5%. It does **not** mean that those electrons are lost, they are simply slightly relocated. Of course, there is no absolute threshold, and it is you who makes the final decision. Nonetheless, a larger charge spilling upon projection translates into less reliable results because the projection does not fully resemble the original wave function. Sometimes, Löwdin's orthonormalization makes these problems disappear almost completely (see our published work on diamond, for example). We urge you, however, to verify this on a case-by-case basis, unless your absolute charge spilling is very small (below 1%). If your calculations yield large charge spilling values, this is the time to reconsider your employed basis functions. Please read through section 1.4 of this manual.

Whenever LOBSTER is not fully convinced that the projected wave function was properly re-orthonormalized, it prints the overlap matrix of the local bands to a file called `bandO-`

`verlaps.lobster`. Ideally, you would expect the latter to be the identity matrix (which is numerically impossible to achieve). If LOBSTER generates `bandOverlaps.lobster`, you should definitely check this file for matrices that deviate strongly from identity (ones on the diagonals and zeroes everywhere else); this points towards serious problems. Note that we print the matrices only if one single entry deviates by more than 0.00001 from the expected value, and this is a rather safe (actually: incredibly safe) criterion. A typical reason for this to happen are columns that are almost zero in the projected coefficient matrix and those point to a less-than ideal choice of the local basis, once again.

4.2 Regarding LOBSTER

4.2.1 Where can I get the newest program version?

The only way to obtain LOBSTER is to visit the official site, www.cohp.de. This way, you can be sure to get the newest version, and no one has tampered with it. Please adhere to the license agreement and do not redistribute the code. If you found LOBSTER anywhere else, please tell us. Please also consider to join the mailing list and send a short message using the subject "subscribe" to cohp-request@lists.rwth-aachen.de. Thank you!

4.2.2 How do I become a legitimate LOBSTER user?

Even though LOBSTER is offered free of charge for academic purposes, **this does not make it "free software"**. The LOBSTER code itself is covered by an all-rights-reserved license and hence you must acquire a valid license in order to execute it legally, by the means stated in the previous paragraph. Note that the LOBSTER license applies to you as an individual person only, so every user is personally required to accept the license agreement. There are no research group or site licenses. In case somebody else introduced you to the software, please take a minute and go to the official website, click on download, fill out the form and accept the agreement. It does not take more than two minutes!

Please understand that we will **not** provide support whatsoever for illicit users. Also, do us a favor and tell us your *real* (and complete) name and *real* (unabbreviated) affiliation because "James Bond" at "MI6" does not do chemical-bonding analysis, at least as far as we know. We guarantee not to share that precious information with anyone.

Nonetheless, if you intend to use LOBSTER for anything not currently covered under the license (e.g. commercial research), please get in touch with us such that we may figure out how to help you!

4.2.3 Which operating systems are supported?

So far, we have tested the following Linux distributions to run LOBSTER successfully:

- Ubuntu 16.04, 18.04, 20.04
- CentOS 5.2, 5.9, 7

Newer revisions of these distributions should also successfully execute LOBSTER, and we expect the same for a large set of distributions besides those listed above. 64-bit versions (also known as amd64 or x86-64) are required, though.

Starting with version 2.0.0, LOBSTER broke the barriers to run under Apple's OS X and the Microsoft Windows platforms. LOBSTER was compiled under OS X 10.11.6 (Yosemite) and using MinGW cross-compiler suite, both in 64-bit mode. Please note that the two provided binaries are not as thoroughly tested as their Linux counterparts, and we cannot guarantee that they will work correctly under all versions and editions of those operating systems, especially not older versions. Still, some of our users might find them useful, even though we strongly recommend using the Linux version, in some cases using a virtual machine running a Linux distribution. Please see www.virtualbox.org how to do so.

4.2.4 Why is the sign of pCOOP and pCOHP different?

By looking at the definitions of COOP and COHP, you will recognize that COOP indicates bonding interactions by a positive sign (gain in overlap), whereas COHP indicates a bonding interaction by a negative sign (lowered energy) – and vice versa for antibonding contributions. To compare these different quantities more easily, one usually *plots* COOP and –COHP because chemists want the bonding levels to go to the right. The same holds for their projected analogues pCOOP and pCOHP. Please have a look at the appendix, it *really* pays off!

4.3 Questions related to the PAW-calculation

4.3.1 How to choose the number of bands?

As said above, you need to set the number of bands at least to the number of basis functions used by LOBSTER. If you do not specify anything else in the `lobsterin`, LOBSTER will build the local atom-centered basis as follows:

- check which orbitals are occupied in the free atom
- for each *l* quantum number, select only those with highest *n*
- We strongly recommend that you always define the local, auxiliary basis explicitly. This is because there can be no “universal”, automatic selection rule for many systems. It simply depends on your chemistry.

Let us look at some examples:

- A `POSCAR` containing two titanium atoms (as in the tutorial files):

  ```
  2×[1×(4s) + 3×(3p) + 5×(3d)] = 2×9 = 18
  ```

  Note that we chose the respective lowest-lying *occupied* orbitals (that is, the 3p, not 4p levels of titanium).

- A cell containing two iron atoms and one nitrogen:

  ```
  2×[1×(4s)+3×(3p)+5×(3d)] + 1×[1×(2s)+3×(2p)] = 2×9 + 4 = 22
  ```

4.3.2 Why do I need so many bands? They are unoccupied anyway!

We know this can be annoying, but the matrix equation that reconstructs the Hamiltonian matrix cannot be solved exactly if the number of bands deviates from the number of basis functions. If there are more bands than basis functions, LOBSTER can and will ignore the additional number of bands and still function properly. But if there are *fewer* bands than basis functions, the equation system becomes under-determined and LOBSTER could only guess the pCOHPs. LOBSTER will continue to run; however, it will give a warning and skip pCOHPs calculations. If you want those, please refer to section 1.3.

4.3.3 Does LOBSTER support spin-orbit coupled calculations?

No, LOBSTER presently does not support spin-orbit coupling (SOC) or non-collinear magnetism (NCM). The reason is that the wavefunctions employ a format that differs heavily from the collinear format, and they would also require different numerical treatment. Nonetheless, we plan at least to implement SOC. If you still want to use LOBSTER for your project relying on SOC or NCM, we suggest you re-calculate your results without those methods and use the new calculations only for LOBSTER analyses. Also, we suspect SOC to be relatively unimportant for chemical-bonding questions, even though it is important for certain physics problems. Here we may be wrong, though.

4.4 VASP-related questions

4.4.1 What is the non-collinear version of VASP?

This is a specially compiled version to deal with non-collinear magnetic moments and such; as a "side product", it also generates **k**-point meshes that cover the entire Brillouin zone (not just the symmetry-inequivalent wedge). The name of this VASP executable can be freely chosen, so we cannot tell you how it is called on your system. Please ask your administrator when in doubt.

4.4.2 So I have to do non-collinear calculations, just for COHPs?

No! It was previously "abused" to generate an `IBZKPT` file that contains no symmetry. There is a much easier way to accomplish the same (see section 1.2 for details). If that strategy does not work for you, however, here is an alternative to make VASP write an `IBZKPT` file for you. (You can also create the full `IBZKPT` file by hand, but this procedure is easier and less error-prone.) Start an `LSORBIT = .TRUE.` calculation using the non-collinear version of VASP and interrupt it (`CTRL+C`) as soon as it writes "`planning FFT`"; there is your `IBZKPT` file! Copy it to `KPOINTS` and have the normal version of VASP run, without the `LSORBIT` flag. You can also reuse that `KPOINTS` file, as long as the number of **k**-points and the cell shape do not change.

4.4.3 Why can't I use the gamma-point version of VASP?

The simplifications applied to the wave functions in VASP (which cause the speedup and memory savings) are not yet implemented into LOBSTER. That means LOBSTER simply cannot digest the output of such a calculation.

4.4.4 What influence does VASP's `LORBIT` tag have?

None. Yes, VASP can carry out a projection onto a local basis on its own; this yields projected DOS in the `DOSCAR` file; nonetheless, these often suffer from a rather large loss in charge. The

LORBIT tag controls the VASP-internal projection scheme. LOBSTER, on the other hand, performs its own projection starting from the wave functions itself and thus does *not* depend on any VASP projection and, hence, it is not influenced by your choice of LORBIT.

4.4.5 Which VASP files will LOBSTER really need?

To digest your VASP data, LOBSTER needs to process the following files: OUTCAR, CONTCAR, POTCAR, WAVECAR, vasprun.xml, KPOINTS, and, in some cases, IBZKPT.

4.5 ABINIT-related questions

4.5.1 Which ABINIT files will LOBSTER really need?

LOBSTER relies on reading the main output file, no matter how you named it. Additionally, it needs to read every .xml file describing the PAW data used in your calculation and the output files with the _EIG and _WFK suffix, but nothing apart from that.

4.5.2 Why can't I use the tetrahedron method in LOBSTER?

In contrast to VASP, ABINIT does not write the table defining the tetrahedra by four **k**-points each to a file (at least to our knowledge), so LOBSTER falls back to Gaussian smearing integration. In case you *do* know how to extract the tetrahedra table from an ABINIT calculation, please get in touch with us.

4.5.3 Why is LOBSTER slow in processing ABINIT data?

The main reason for that impression has most likely been answered before: the implementation of Gaussian smearing integration in LOBSTER is slower than using the tetrahedron method. Apart from that, VASP lists the reciprocal projector functions in their PAW data, while ABINIT transforms their real-space representation on-the-fly. Hence, LOBSTER must do the same and this transformation takes (a little) time. At the moment, there is not much we can do about that.

4.5.4 Which ABINIT versions are supported?

Since LOBSTER can only read PAW-XML data sets, ABINIT 7.6 is the minimum requirement. Versions 7.8 and 7.10 are known to work, too, and as long as there won't be any changes to the format of ABINIT output files we safely assume that future versions of ABINIT will be supported as well.

4.5.5 Why are the `lobsterin` files for the ABINIT examples different?

ABINIT and VASP employ different PAW datasets and, hence, are subject to different electronic (valence) configurations. Of course, these differences are also mirrored in the choice of the local basis, and this is the reason for the necessary differences.

4.6 Quantum ESPRESSO-related questions

4.6.1 Which Quantum ESPRESSO files will LOBSTER really need?

To do the projection, first of all LOBSTER needs to read the main input file carrying the `.scf.in` suffix to extract some basic information of the crystal structure under question. Additionally, LOBSTER needs to read every `.UPF` file describing the PAW data used in your calculation and the output files which are all stored in a directory that carries the `.save` suffix. Since LOBSTER 5.0, Quantum ESPRESSO's hdf5 file format is supported as well.

4.7 Technical trouble

4.7.1 LOBSTER crashes due to unknown symbols. What now?

The newer Linux versions are statically linked binaries, so there should not be any symbol lookups at all. If you still get that problem, first *make sure* you are actually running the newest version and did not accidentally mix it up with an old version. If the problem persists, please *make sure* that you have installed all available updates for your Linux distribution. If it still fails, your Linux distribution itself might be too old and is thus unsupported (please see the previous section). If that happens on OS X or Windows, feel free to contact us.

4.7.2 Why do results differ with a newer version of LOBSTER?

For a newer version of LOBSTER, all code changes have been documented as transparently as possible in the `CHANGELOG.PDF` file included in the distribution package. In case you end up with differing results, please first make sure that the results not only differ with respect to numerical accuracy (note that the `WAVECAR` file only stores single precision!). Please keep in mind

that LOBSTER relies on (compile-time) libraries, so changes therein can influence LOBSTER results, too. If you are convinced that something is wrong, however, please contact us.
Version 2.0.0, on the other hand, changes the program's default behavior by employing different (i.e., *better*) algorithms at certain stages of the calculation. Such deviations to previous LOBSTER versions are intentional improvements. Hence, we clearly recommend to only use the backwards compatibility functions for, say, historical reasons or consistency tests.

4.7.3 The parallel version does not scale ideally! Why?

First, ideal scaling is almost impossible to achieve in the real world, and we are convinced that getting the results faster by "only" one order of magnitude despite using 12 CPUs is still much better than going serial. Second, if you cannot resist benchmarking LOBSTER, please keep in mind that almost all modern CPUs have a so-called "turbo" function. The latter may increase the clock frequency of some cores quite heavily in case not all CPU cores are used. As a consequence, a CPU nowadays provides more FLOPS per core for serial programs than for parallel ones, and that may be a reason for missing speed-up.

Please also note that some CPUs announce more "logical" cores to the operating system than they physically have. This is especially true on Intel CPUs that support Hyperthreading™ and AMD CPUs featuring the "module" concept. So if you observe a nice speed-up for the first half number of cores but then stagnation, this is likely caused by the hardware.

Even though LOBSTER has been carefully optimized, it might still not be fast enough to get your results in a reasonable time. If that is the case, please contact us. However, we may suggest using a smaller but chemically still useful model.

4.7.4 Why does LOBSTER get killed before it is done?

Given certain circumstances your LOBSTER calculation might stop prematurely printing a message similar to the following:

```
projecting...
0%   10   20   30   40   50   60   70   80   90   100%
|----|----|----|----|----|----|----|----|----|----|
Killed
```

This means that your operating system decided to abort the execution of LOBSTER. Most probably your machine just ran out of memory, but there are other possible reasons as well. Under Linux you can run the following command to see for yourself:

```
dmesg | tail
```

You can try to stop other running processes and try again. In most cases, however, you will have to move to a bigger machine to finish the calculation successfully.

4.7.5 Why did I get negative spilling values?

While spilling is bounded by zero in theory, numerically approximating algebraic expressions introduce deviations; possibly even *over*estimating the result. Additionally, some of the overlap integrals LOBSTER calculates are not bounded to unity, simply due to PAW theory. Then, error propagation can (and, in cases, does) make the norm of the projected wavefunction exceed unity and, thus, causes negative spilling. Furthermore, while augmentation spheres are formally not allowed to overlap, they often do so in computational practice—as always, trading accuracy against faster calculations. As LOBSTER does not cope with such overlap explicitly, this might also lead to deviations from the "ideal" behavior.

Fortunately, slight deviations of this kind are often taken care of using Löwdin's orthonormalization, so final results can still be fine. Please, as always, do carefully verify your output in any case.

However, since LOBSTER only prints an average of the spillings over all **k**-points and bands, it is very well possible that negative and positive spilling values cancel each other. Consequently, it could happen that the spilling appears to be better than the projection actually is. To counteract, we now take the absolute value of the spillings of all bands before averaging. This certainly weakens the displayed percentage of lost electron density, but it gives a real "smaller is better" criterion in return.

4.7.6 Why does the cohpGenerator feature not find certain bonds?

Since LOBSTER 3.0, a new algorithm has been implemented to allow LOBSTER to automatically calculate interactions of bonds forming with neighboring cells using only the primitive unit cell given that the bond lengths are within the specified range. This question should no longer pop up for LOBSTER 3.0, so please make sure to use the latest version.

4.8 And what if something does not work?

First of all, we did not give you a guarantee it would. We can also not guarantee that it will solve your scientific problems. Sadly, software is almost never free of errors, and we cannot prepare it for any possible scenario, albeit we do our best. So, if something does not work as it is supposed to, please do **file a bug report** (see below). We keep an automatic tracking system for such reports, and you can choose to be notified as soon as this particular issue is resolved. The same goes for features you would like to see in the software or anything (really, anything) that you think can or should be improved. Remember: We can only make LOBSTER better if we know what you need, and we can only repair errors about which we know. Do not be shy to be an active user. **Thank you for your support!**

5 Contact and Bug Reports

For technical questions apart from that, feel free to send an email to mail@cohp.de but please do not expect an answer within one hour. To discuss chemistry, contact the Dronskowski group via the usual way (www.ssc.rwth-aachen.de). To make sure that you will receive information about future releases, please join the mailing list and send a short message using the subject “subscribe” to cohp-request@lists.rwth-aachen.de.

6 Version history

1.0.0 (October 2013)	• First LOBSTER version released to the public!
1.0.1 (November 2013)	• The **k**-points are now read from `IBZKPT` (instead of `vasprun.xml)` for higher accuracy. If your results differ from those of 1.0.0, this should be the reason. • The latter is also done for ion positions using `CONTCAR`. • Fixed problems caused by VASP 4.6 writing a different value for the number of bands than it has actually used to `vasprun.xml`. • Fixed `cohpgenerator` ignoring different atomic species (`type A type B`)
1.1.0 (February 2014)	• algorithmic changes yielding notable (and serial) performance improvements • reduced memory usage • parallelism using OpenMP • basic support for `_sv` and `_pv` potentials in VASP • GSL is not used anymore! This enables static linking and should also solve the problems with unknown symbols (`_ZGVNSt7collateIwE2idE`) on old Linux distributions. • several minor bugfixes • VASP does not (over-)write `IBZKPT` if a non-automatic **k**- point generation scheme was requested in `KPOINTS`. LOBSTER now checks for this and reacts accordingly. • If there are more PAW bands than local basis functions, LOBSTER now ignores the excess bands by default. If you insist for whatever reason, you may get the old behavior back by adding `doNotIgnoreExcessiveBands` to `lobsterin`. With this fix there is no need to set `NPAR = 1` in VASP anymore.

1.2.0 (March 2015)	• updated build-toolchain (GCC 4.8.4, Eigen 3.2.4, boost 1.57) • fixed: a segmentation fault occurred if certain VASP-output files were empty • fixed: in rare cases VASP does not write the Fermi level to vasprun.xml and then LOBSTER did not read it correctly • improved exception handling • improved file and console I/O • fixed: crashes due to missing f-orbital name-strings • improved speed of PW-system initialization, especially for large systems • fixed: spilling showed NaN if not a single band is occupied • fixed: `lobsterin` files containing MS-DOS-type linebreaks caused crashes • fixed: the calculation of wavefunction characters included an erroneously twofold conjugation. This severe bug caused spilling values (of p and f orbitals) to be too high and given certain circumstances impaired the results of the projection. But in many cases the errors cancel out widely. • fixed: The Zr_sv POTCAR could not be read because it contains a typo • fixed: Combining two POTCARs, where one element symbol is part of the other in a certain order (e.g. O, then Co), were read improperly. • fixed: wrong value for IpCOHP of the second spin channel was printed to the ICOHPLIST.lobster file when using the orbitalwise keyword • added improved basis set (`pbeVaspFit2015`) • added functionality to print the employed basis functions • added support to read `WAVECAR` files in big-Endian format (e.g., PowerPC) • many code-internal improvements

2.0.0 (December 2015)	• new: ABINIT support • new: BWDF feature • new: basis function rotation feature • new: detection of symmetry-equivalent bonds, so pCOHP is printed per bond instead of per cell, which is more consistent with respect to supercells • improved Hamiltonian matrix reconstruction • improved absolute spilling criterion • improved parallelization • fixed: Gaussian-smearing method used a wrong stepWidth to calculate pDOS, pCOOP and pCOHP values. Integrated values were not affected. • updated build-toolchain (GCC 5.2.0, Eigen 3.2.7, boost 1.59) • supporting Apple OS X and Microsoft Windows • many code-internal improvements
2.1.0 (September 2016)	• new: saving projection results to a binary file for later reuse • new: automatic alignment of individual basis functions • new: fatband feature • new: RMSp feature • new: BWDF can be weighted with integrated pCOHPs as well • new: writing transfer, overlap, coefficient and Hamiltonian matrices to file • new: real-space Hamiltonian and Overlap matrix calculation • new: wavefunction plotting feature • fixed: calling COHPGenerator with default parameters did not find any interactions • fixed: unit cells defined using negative values led to wrong bond lengths • fixed: typos in f-orbital basis rotation matrix • improved vectorization • improved memory management for projectors • updated build-toolchain (GCC 5.4.0, Eigen 3.3.0-beta2, boost 1.61), use LTO

2.2.0 (October 2017)	• new: DOE feature • new: Quantum ESPRESSO support • new: skipping-writing-output feature • fixed: ABINIT recently modified its _WFK file format in its new version (≥ 8.0) • fixed: orbitalwise-ICOHP outputs give wrong values due to wrong column chosen from COHP table to write-out • fixed: nBands limited by the number of band of PAWCrystal class created error • fixed: some JTH-xml potential of ABINIT does not have the pw-ecut information • improved flexibility in defining energy range of COHP after saving the projection. • improved algorithm for automatic orbital rotation • improved format of wavefunction files by inserting a header to enable 3D-plotting with wxdragon
2.2.1 (November 2017)	• fixed: a VASP routine adding eigenvalues and occupation numbers could result in a race condition if run in parallel • fixed: a QE routine had a problem reading one-letter atom symbols in AFM setups • fixed: a QE routine calculating unit-cell volumes could result in negative values causing NaN output • improved fatband feature by allowing multiple calculations in a single go. Furthermore, only **k**-points associated with the path of the bandstructure are printed

3.0.0 (August 2018)	• new: Mulliken and Löwdin population analysis (output files are `CHARGE.lobster` and `GROSSPOP.lobster`) • fixed: QE changed its output formats since version ≥ 6.0 • fixed: ABINIT **k**-points weights were not properly normalized • fixed: ABINIT sometimes doesn't output the Fermi energy so a function was added to get the Fermi level from some other output file • fixed: a bug which made the ABINIT port unable to detect the atom positions without the keyword "xred" • improved algorithm to calculate COHP, COOP and their integrated value more precisely and to allow interaction calculations with neighboring cells using only the primitive unit cell • improved algorithm to calculate the Hamiltonian and ensure its hermiticity
3.1.0 (December 2018)	• improved the speed of the real-space-based COOP/COHP calculations by parallelizing the Fourier transform parts • new: basis-functions recommendation feature based on pseudopotential files or atomic valence configurations • new: skip-projection feature • fixed: bugs in the QE port which mistakenly read the "alat" variable and number of bands
3.2.0 (May 2019)	• algorithmic changes in fatband to make it consistent with DOS • modified indices for the orbitalwise interactions to be consistent with the total interaction indices • modified the format for outputs with a matrix form • setup `GROSSPOP.lobster` as a default output • new: skipGrossPopulation keyword • new: **k**-resolved COHP feature • fixed: bugs in 5f basis functions of Ac & Th which caused an error in the orbital-overlap calculations

4.0.0 (June 2020)	• algorithmic changes in all chemical-bonding routines to accommodate time-reversal symmetry which speeds up the projection by an average factor of two • required files for VASP simplified • modified the format of bandOverlaps.lobster so that the spin indices follow the regular convention, (1&2) instead of (0&1) • added the "to" coordinate points in 3D wavefunctions files • added cartesian coordinates of **k**-points in kspaceCOHP files • added a progress bar for DOS, COOP, and COHP in lobsterout • new: "around" keyword to further filter cohpGenerator • fixed: a bug in writing real-space matrices causing the output to be identity matrices all the time unless the keyword "doNotOrthogonalizeBasis" is used • fixed: bugs causing an error in reading QE pseudopotential files with double quotes from the input file • fixed: bugs causing an error in reading parameters in QE > 6.0 • fixed: a bug causing an error when a user chose Bunge for elements with $Z > 54$; LOBSTER now automatically switches from Bunge to Koga in this case. • fixed: bugs causing PAW and LCAO wavefunctions not symmetric when printed
4.1.0 (August 2021)	• modified the columns (positions) for the "from" and "to" atoms in ICOOP & ICOHP as well as COOP & COHP to be consistent with visualization using wxDragon. • added orbitalwise output to ICOHPLIST.lobster and ICOOPLIST.lobster files • new: Crystal Orbital Bond Index (COBI) for two-center and multi-center interactions • new: Ewald method for calculating electrostatic lattice potentials and Madelung energies • new: DOS from orthonormalized basis • fixed: bugs causing false rotations of d orbitals.

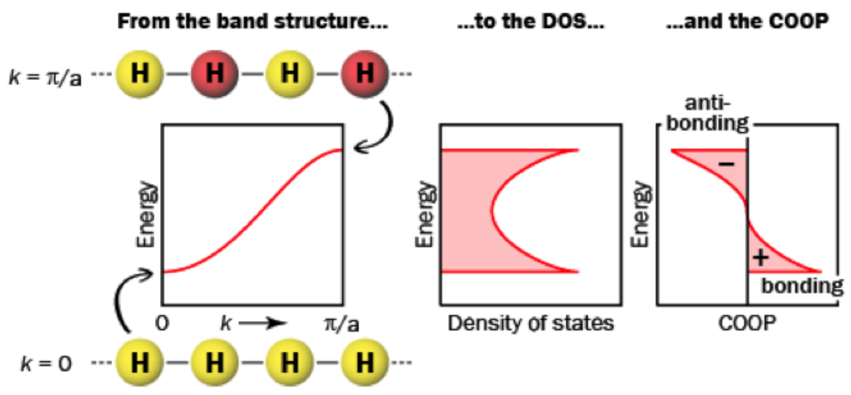
5.0.0 (September 2023)	• added output file "POSCAR.lobster" containing the atomistic structure in POSCAR file format • added support for Quantum ESPRESSO's hdf5 file format output (Linux only) • changed evaluation of ICOOP/ICOHP/ICOBI to fully analytic calculation, not relying on the chosen energy range • new: polarization obtained from Mulliken and Löwdin charges • new: Embedded Localized Molecular Orbitals • new: flexible STO basis input to add arbitrary basis functions • new: generic electronic-structure input to support various electronic-structure codes (Linux only) • fixed: bug causing incorrect electronic-structure projection when starting from newer VASP POTCAR file formats • fixed: bug where non-existing atom indices in multi-center interaction inputs caused LOBSTER to crash • fixed: bug leading to inconsistencies between NcCOBILIST and ICOBILIST outputs • fixed: bug in output of spin-polarized NcICOBI data

5.1.0 (June 2024)	• new: Linear Combination of Fragment Orbitals (LCFO) method: – Canonical molecular orbitals as cube files – MO diagrams for fragment orbitals – DOS, COHP, COBI, Löwdin, fatband analyses for fragment orbitals – Molecular orbital formation energy (MOFE) as a new method to quantify energetic contributions of atomic orbitals to molecular orbitals • new: added function to write electron density of atomic orbitals • new: improved output of ICOOP, ICOHP, ICOBI and GROSSPOP files for spin-polarized calculations • new: `skipCar` keyword • new: made analytical calculation of integrated COOP, COHP and COBI list files as default • fixed: output of `writeBasisFunctions` to be consistent with `custom` basis • fixed: added OpenMP parallelization to basis set rotation • fixed: added keyword `useDecimalPlaces` to `CHARGE.lobster` • fixed: renamed `POSCAR.lobster` to `POSCAR.lobster.vasp` to improve compatibility with visualization programs that support the VASP file format • fixed: reduced memory consumption of potential reconstruction routine used in embedded localized molecular orbital feature
5.1.1 (October 2024)	• fixed: bug that caused analytical ICOBI calculation be off by a factor of 2 • fixed: bug that caused a segmentation fault for orbitalwise ICOHP and ICOBI in the LCFO basis

Appendix: Chemical Bonding in a Nutshell

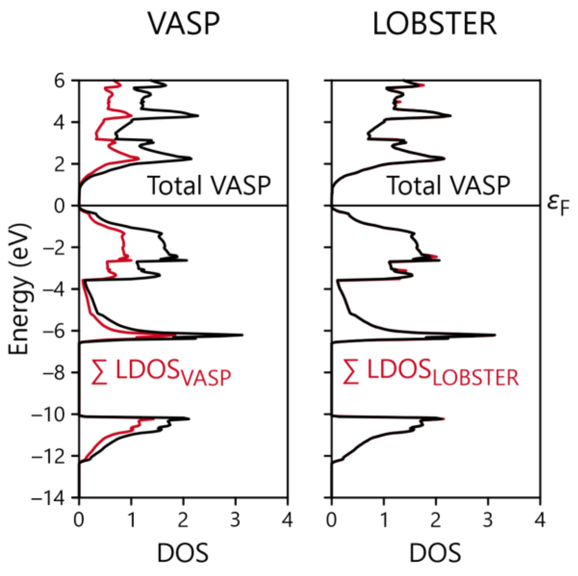
No, this is *not* a complete account of chemical-bonding theory. (Did we hear a sigh of relief?) Instead, we summarize the main ideas behind LOBSTER here as a first, quick introduction. A more detailed version of this, along with much other information, is available on our website www.cohp.de. We have also added links to some of the original literature; do not hesitate to look at those papers, too!

The COHP (or COOP) concept is most easily understood by looking at the simple band structure of a "one-dimensional" solid; the following example is from a classic introduction. Imagine a linear chain of hydrogen atoms, the one-dimensionally periodic analogue of H_2 (whose molecular-orbital scheme is known from the freshmen lecture). Along this chain, each H atom carries one 1s atomic orbital, and we may sketch them in the following figure by colored spheres. The ideal combination of these 1s orbitals would certainly be one where all have the same plus/minus sign (visualized by like coloring); this is the H–H bonding (in-phase) crystal orbital, characterized by a positive overlap population. At high energies, we find alternating plus/minus signs between neighboring atomic orbitals; this is the H–H antibonding (out-of-phase) combination with a negative overlap population:



The density-of-states (DOS) diagram in the middle results as the inverse slope of the band structure on the left. Note the plotting style which does not make sense from a physicist's perspective (the DOS is a function of the energy, not vice versa): chemists want to see how the DOS emerges from the bands; it's just a matter of cultures.

Before we dive deeper into chemical-bonding analysis, let's first reconsider certain aspects of different DOS projection schemes. Strictly speaking, a decomposition of the total electronic band structure or DOS into atomic (local) contributions is not restricted to a local-orbital basis set and can – in principle – also be done within a PW/PAW framework, even though it then remains somewhat qualitative in nature. For clarity, let's compare the projected DOS from VASP and LOBSTER for gallium arsenide below:



On the left, there is the total DOS of GaAs (in black) as accurately calculated from VASP, and the sum of the local DOS contributions of Ga and As (in red) almost coincides with the total DOS, despite the fact that about 25% of the charge (the valence electrons) gets lost by the qualitative VASP projection. When projecting to true atomic orbitals via LOBSTER (right picture), the total DOS by VASP and the sum of the local DOS by LOBSTER are practically identical such that exactly 8 valence electrons (3 for Ga plus 5 for As) are recovered.

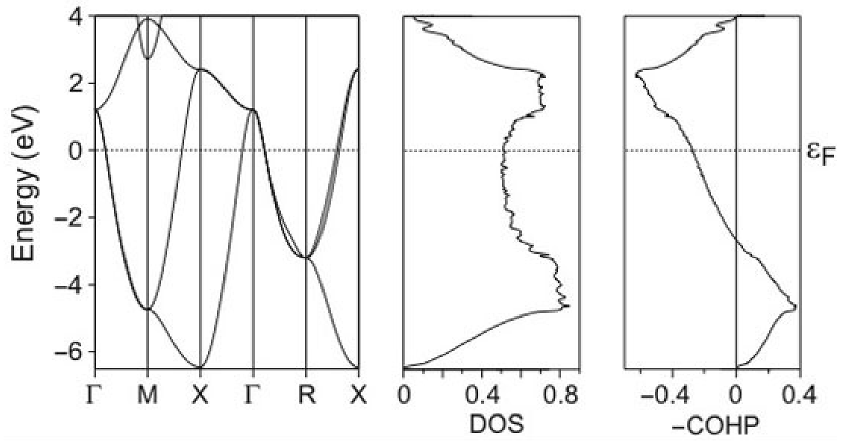
The COOP indicator results from multiplying the DOS by the overlap population (hence its name), and it adds an additional dimension—the precious bonding information: COOP adopts positive values (bonding, because of the positive overlap population) in the lower region of the band, and negative values (which identify antibonding interactions) at higher energy levels. By comparing the band structure and its orbital icons with the COOP diagram, it is obvious why the nearest-neighbor COOP is bonding at low energies and antibonding at high energies.

If one can weight the DOS by the overlap population, one can also weight it by the corresponding element of the Hamiltonian. In fact, good reasons within density-functional theory (variational in terms of the Hamiltonian, not the overlap) suggest to choose the latter method, dubbed crystal orbital Hamilton population (COHP). By doing so, we partition the band-structure energy (instead of the electrons) but again into bonding, nonbonding, and antibonding contributions. There is also a difference in the plus/minus signs: we recall that bonding is characterized by a *positive* overlap population; the corresponding Hamiltonian off-site element (physicists would probably call it "hopping term") will then be *negative*, just like the H_2 molecule *lowers* its energy by forming a bond! Consequently, antibonding interactions exhibit positive off-site Hamiltonian elements.

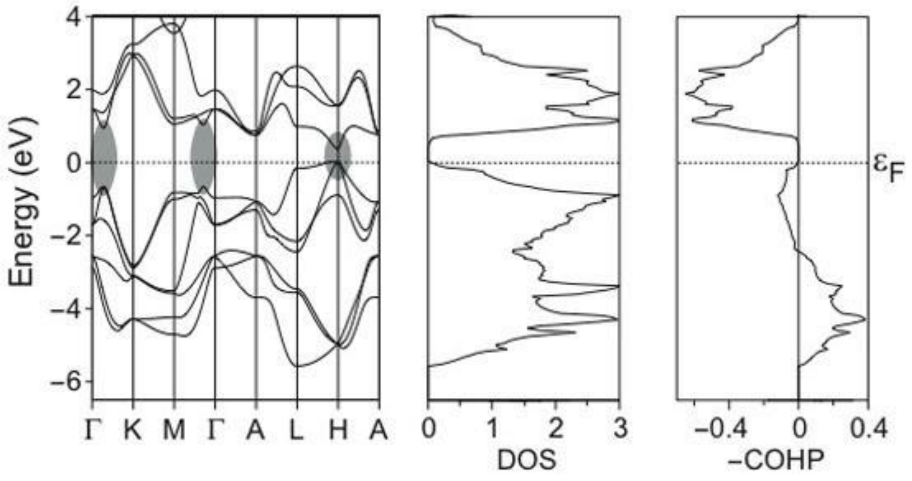
To make COHP plots look similar to COOP, most people (including us) are plotting –COHP diagrams such that bonding states (positive, to the right) and antibonding states (negative, to the left) are easy to grasp. Note that there is not a *direct* equivalence between COOP and –COHP.

And yet, there is another way to look at things. In 2021, the crystal orbital bond index (COBI) was introduced, a generalization of the Wiberg/Mayer bond index for the translationally invariant solid. COBI plots are analogous to COOP/−COHP plots but offer a new perspective: the integrated value (the ICOBI) equals the chemical bond order, for example ICOBI = 3.00 for the triple bond in the dinitrogen molecule, so COBI is extremely simple to understand and should serve well for chemical-bonding analysis in solids, too.

Let us look at a more realistic example. This finger exercise, published 2002, nicely illustrates how nature optimizes structures by the above principle of maximum bonding. If one calculates the band structure of, say, elemental tellurium assuming a primitive, simple cubic crystal structure, one arrives at the following bands, DOS and COHP:

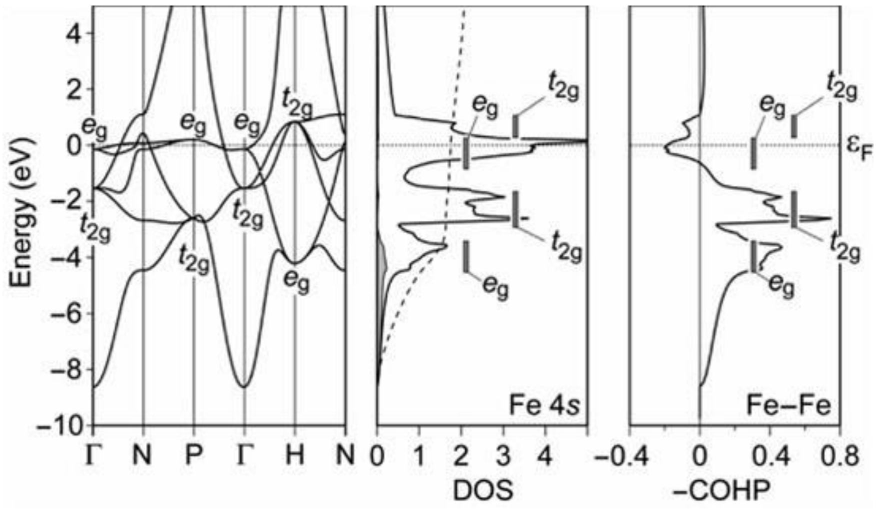


Some bands cross the Fermi level, leading to a significant DOS; cubic Te would be expected to be an electric conductor. Te–Te bonding, however, is clearly unoptimized since there are populated (!) antibonding states which lead to an enormous internal stress. Just like in molecular orbital theory, the antibonding states at high energies reflect an electronic instability: if left alone, cubic Te immediately “explodes” and adopts its famous ground-state structure containing helices of Te atoms. You may now guess how the bands, DOS and COHP look like; here they are:

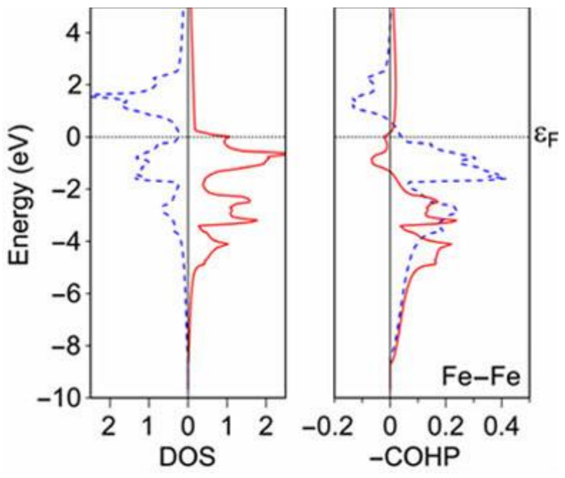


Upon structural change, a band gap opens up, depleting antibonding areas; indeed, hexagonal Te is a small-band-gap semiconductor, as observed in the DOS. Focusing on the COHP plots, the antibonding interactions below the Fermi level have become much smaller. Again, nature optimizes bonding.

This approach can be further extended to spin-related phenomena, offering a new look at itinerant magnetism of the transition metals. We would first like to know how the band structure of body-centered cubic (bcc) iron looks like without spin-polarization. In other words, we do a *gedankenexperiment* on an Fe phase which does not exist at all, namely nonmagnetic iron at zero Kelvin—here are the bands, the DOS and the Fe–Fe COHP for nonmagnetic Fe, that is, with all spins paired:

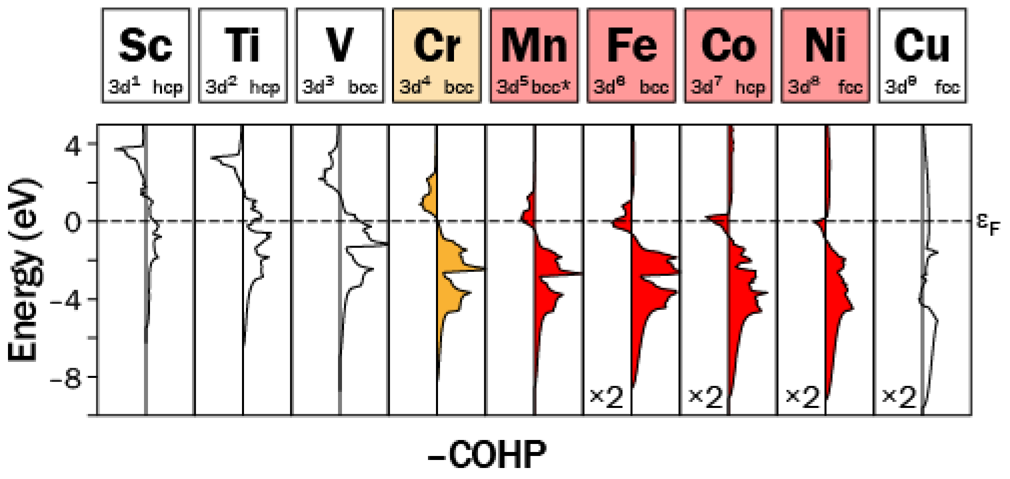


The bands in the valence region are mostly 3d, naturally, and there results the typical three-peaked DOS of a bcc transition metal. The curious thing about the nonmagnetic COHP is that the Fermi level falls in a strong Fe–Fe *antibonding* region. Again, we have discerned an electronic instability! Consequently, one would expect some kind of structural change—which here does not occur: bcc-Fe stays bcc-Fe. The answer to that puzzle is that nonmagnetic iron *does* undergo a distortion, but instead of the atoms rearranging themselves, the electrons do. Nonmagnetic bcc-Fe is unstable with respect to an *electronic structure distortion*, which makes the two spin sublattices inequivalent, thereby lowering the energy and giving rise, upon ordering, to ferromagnetism; these are the spin-polarized results:



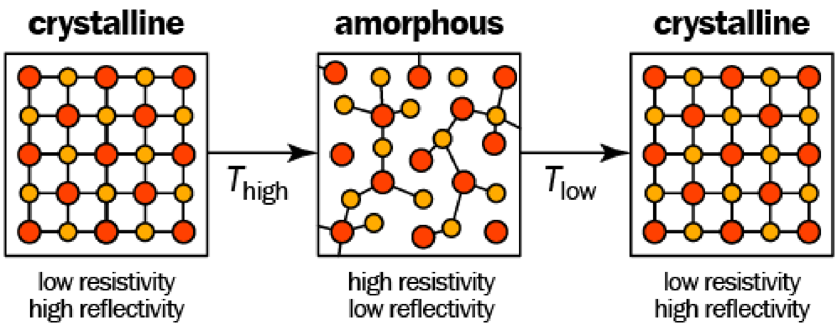
Upon spin-polarization, the total energy lowers by about 0.43 eV, paralleled by a strengthening of the Fe–Fe bonds by roughly 5 per cent. The shifts in the majority (red) and minority (blue) spin sublattices have removed the antibonding states at the Fermi level, thereby maximizing the Fe–Fe bonding as far as possible. It looks like a new, purely electronic kind of Jahn-Teller distortion since only the electrons are involved in breaking the symmetry.

What about the other transition metals? The following picture shows the nonmagnetic DOS and metal-metal COHPs for the entire 3d row:

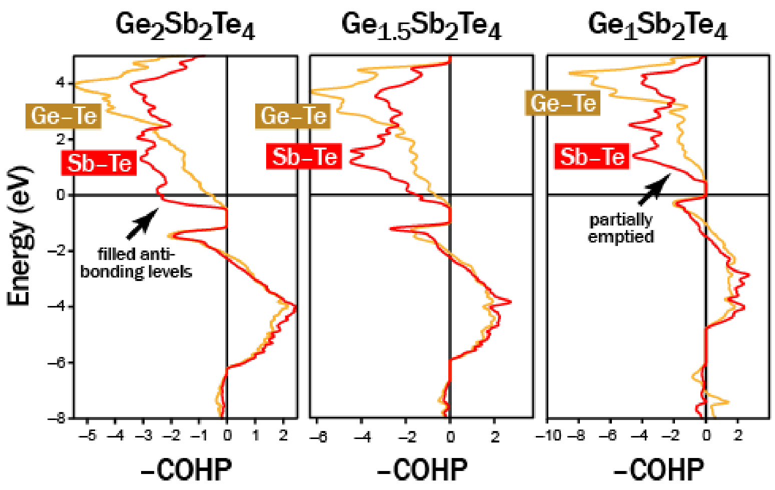


For the early transition metals like Ti, the Fermi level lies low in the COHP curve and thereby falls in the metal–metal bonding region: there is no drive towards ferromagnetism for the early transition metals. For Cr (a typical antiferromagnet), the Fermi level lies in the COHP curve between the bonding and antibonding regions, whereas for the metals from Mn (which we describe in a bcc structure for simplicity) to Ni, the Fermi level lies in the clearly antibonding region; Fe, Co, and Ni are all ferromagnetic. An extension of this idea to more complicated alloys is straightforward: if you want to synthesize a new ferromagnet, try to adjust (by chemical means such as oxidation and reduction) the Fermi energy of the material such as to push it into antibonding interactions. We may refer the reader to some of the original literature.

Next, let us look at a totally different field of chemistry, in which COHP analysis has been quite useful as well: that of data-storage materials (the ingredients for computer memories; yes, these are made by solid-state physicists and chemists). To encode “one” and “zero” bits, there are so-called phase-change materials (PCMs) which can be switched between two different solid-state forms, usually a crystalline and an amorphous phase. Crudely sketched, it looks like this:

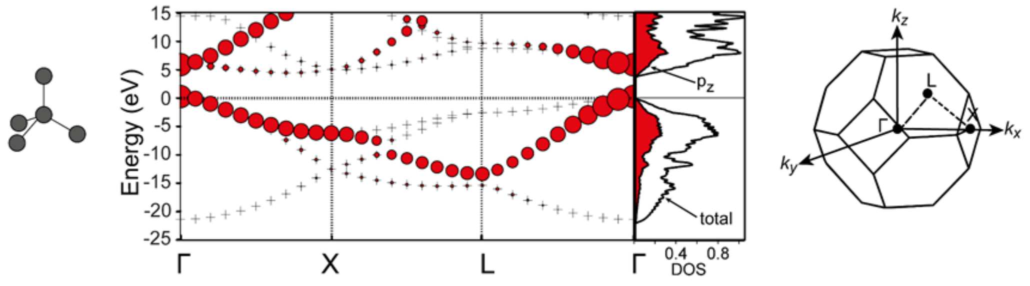


Many such PCMs are found in the ternary germanium–antimony–tellurium system. $Ge_2Sb_2Te_5$, for example, is found in DVD-RAM disks whereas $Ge_8Sb_2Te_{11}$ is employed in the re-writeable Blu-Ray disk. All "GST" materials are closely related chemically, no matter the exact stoichiometric composition. Thus, we have calculated COHP curves, starting with the hypothetical compound $Ge_2Sb_2Te_4$ (left), taking the rocksalt structure with Ge and Sb randomly distributed on the cationic sublattice, and Te fully occupying the other. Indeed, this very compound does *not* exist; surprisingly only at first sight, it decomposes by expelling pure germanium.



The above picture shows why. Both Ge–Te and Sb–Te bonding clearly falls into antibonding regions around the Fermi level if one starts with the fully occupied, rocksaltlike "224" phase. By removing cations (and creating vacancies on the corresponding sublattice), the Fermi level is lowered, removing electrons from these unfavorable areas. On the right, the GST-124 material is characterized where a noticeable part of these antibonding levels has been emptied, and the remaining structure is strengthened this way. Please, as always, have a look at the original story for a more detailed presentation as possible here.

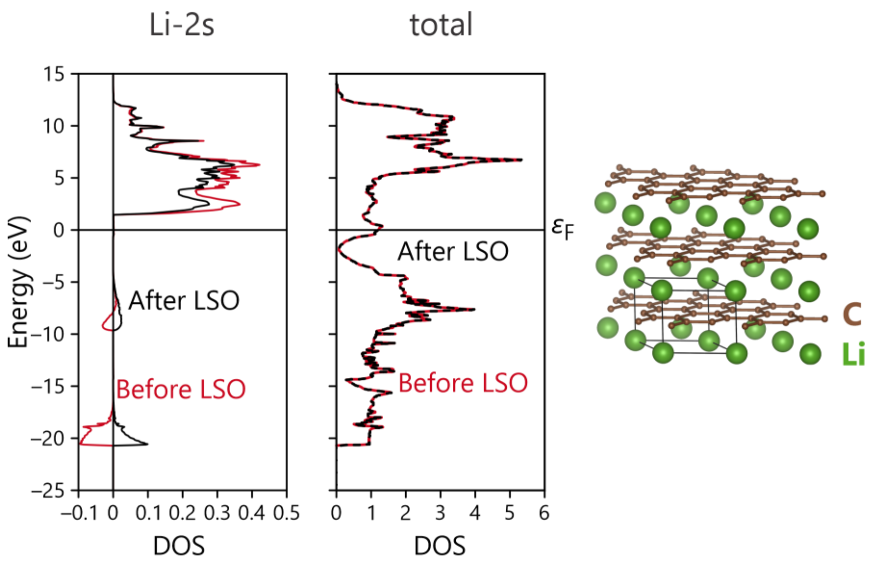
And yet, there is even more. Because LOBSTER reconstructs the entire plane-wave electronic structure using local orbitals, tried-and-true solid-state physics techniques previously available using LMTO theory may also be carried out using plane waves. We all know that it's possible to project local DOS (say, the Ga contribution of the DOS of GaAs) besides the total DOS using a fine electronic-structure program. In addition, LOBSTER will not only allow to project, say, the p_z DOS contribution of the carbon atom in diamond, it will also decorate the bands of diamond using the famous *fatband* technique. That is to say that each band in reciprocal space is graphically fattened according to the amount of the p_z contribution to that band at each **k**-point:



The total and p_z DOS already show how p_z mixes into the electronic structure; the fatband plot further visualizes that the red band below (and above) the Fermi level is relatively "pure" p_z between Γ and X, and also between L and Γ, but less so (a weaker contribution) between X and L. We have provided many more examples in the original fatband publication. In addition, one may construct any linear mixing DOS (such as sp^2 etc.), and LOBSTER will perform automatic orbital rotations to match the local structural motifs. Again, we invite you to study the original papers.

Before finalizing this appendix, let's reiterate some DOS technicalities. As already mentioned, the local DOS as exactly projected from an LCAO basis set have a quantitative, not only qualitative meaning. But even within the LCAO framework, there are different ways to set up the DOS, so LOBSTER provides two such approaches. The COOP as well as the Mulliken gross populations and charges are calculated from a *non-orthonormal* basis set. In order to achieve projected COHP that are comparable to traditional COHP, the local basis is *orthonormalized* using Löwdin's symmetrical orthonormalization (LSO), and this orthonormal basis set is then used for calculating Löwdin gross populations and charges as well as the COBI.

In general, the DOS as calculated from both approaches will not differ much as only some orbital contributions become slightly altered. However, the idiosyncrasy of PAW theory and some of its pseudopotentials may indeed lead to *negative* DOS as well as negative gross populations and thereby unphysical charges, usually encountered for elements with very few valence electrons (say, alkali metals). Here, by applying Löwdin's symmetrical orthonormalization (LSO) to the local basis, this problem can be solved, as mirrored from the LiC_6 example below showing significant negative DOS for lithium's 2s orbital using the non-orthonormal basis but not so for the orthonormalized basis. At the same time, the compound's total DOS remains the same because orthogonalization only affects DOS partitioning. Please keep in mind that LSO does not change anything related to COOP or Mulliken gross population and charges, and thus they are still invalid if the original DOS is negative. Powerful alternatives (COHP, Löwdin charges) are available, though.



In case you want to know more about chemical-bonding in general, consider looking into this book, light in style, written for newcomers of chemical bonding, with plenty of examples: