USPEX Computational Materials Discovery

Universal Structure Predictor: Evolutionary Xtallography

A.R. Oganov, C.W. Glass, A.O. Lyakhov, Q. Zhu, G.-R. Qian, H.T. Stokes, P. Bushlanov, Z. Allahyari, P. Graf, V. Stevanovic, F. Therrien, S. Lepeshkin, Z.H. Wang,

with contributions from
M. Davari, A. Samtsevich, R. Agarwal, X. Dong, M.S. Rakitin, P. Pertierra, Z. Raza, M.A. Salvado, D. Dong, Q. Zeng

MANUAL

Version 10.2, January 16, 2019.

http://uspex-team.org

Oganov's Lab

Computational Materials Discovery Laboratory

Contents

1 Features, aims and history of USPEX

1.1 Overview

USPEX stands for *Universal Structure Predictor: Evolutionary Xtallography*... and in Russian "uspekh" means "success", which is appropriate given the high success rate and many useful results produced by this method! The USPEX code possesses many unique capabilities for computational materials discovery. Here is a list of features: ...

From the beginning in 2004, non-empirical crystal structure prediction was the main aim of the USPEX project. In addition to this, USPEX also allows one to predict a large set of robust metastable structures and perform several types of simulations using various degrees of prior knowledge. Starting from 2010, our code explosively expanded to other types of problems, and from 2012 includes many complementary methods.

The problem of crystal structure prediction is very old and does, in fact, constitute the central problem of theoretical crystal chemistry. In 1988 John Maddox[1] wrote that:

> "One of the continuing scandals in the physical sciences is that it remains in general impossible to predict the structure of even the simplest crystalline solids from a knowledge of their chemical composition... Solids such as crystalline water (ice) are still thought to lie beyond mortals' ken".

It is immediately clear that the problem at hand is that of global optimization, *i.e.*, finding the global minimum of the free energy of the crystal (per mole) with respect to variations of the structure. To get some feeling of the number of possible structures, let us consider a simplified case of a fixed cubic cell with volume V, within which one has to position N identical atoms. For further simplification let us assume that atoms can only take discrete positions on the nodes of a grid with resolution δ. This discretization makes the number of combinations of atomic coordinates C finite:

$$C = \frac{1}{(V/\delta^3)} \frac{(V/\delta^3)!}{[(V/\delta^3) - N]!N!} \tag{1}$$

If δ is chosen to be a significant fraction of the characteristic bond length (e.g., $\delta = 1$ Å), the number of combinations given by eq. 1 would be a reasonable estimate of the number of local minima of the free energy. If there are more than one type of atoms, the number of different structures significantly increases. Assuming a typical atomic volume $\sim$10 Å^3, and taking into account Stirling's formula ($n! \approx \sqrt{2\pi n}(n/e)^n$), the number of possible structures for an element A (compound AB) is 10^{11} (10^{14}) for a system with 10 atoms in the unit cell, 10^{25} (10^{30}) for a system with 20 atoms in the cell, and 10^{39} (10^{47}) for a system with 30 atoms in the unit cell. These numbers are enormous and practically impossible to deal with even for small systems with a total number of atoms $N \sim 10$.

Even worse, complexity increases exponentially with N. It is clear then, that point-by-point exploration of the free energy surface going through all possible structures is not feasible, except for the simplest systems with $\sim$1-5 atoms in the unit cell.

USPEX[2;3] employs an evolutionary algorithm devised by A.R. Oganov and C.W. Glass, with major subsequent contributions by A.O. Lyakhov and Q. Zhu. Its efficiency draws from the carefully designed variation operators, while its reliability is largely due to the use of state-of-the-art *ab initio* simulations inside the evolutionary algorithm. The strength of evolutionary simulations is that they do not require any system-specific knowledge (except the chemical composition) and are self-improving, i.e. in subsequent generations increasingly good structures are found and used to generate new structures. This allows a "zooming in" on promising regions of the energy (or property) landscape (Fig. 1). Furthermore, by carefully designing variation operators, it is very easy to incorporate additional features into an evolutionary algorithm.

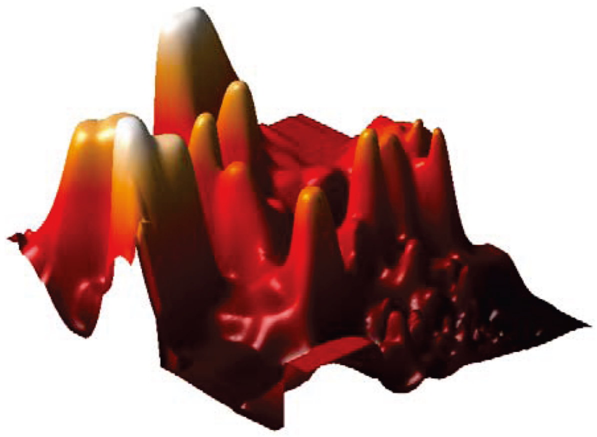


Figure 1: **2D projection of the reduced landscape of Au_8Pd_4, showing clustering of low-energy structures in one region.** The landscape was produced using the method of Oganov & Valle (2009).

A major motivation for the development of USPEX was the discovery of the post-perovskite phase of $MgSiO_3$ (Fig. 2), which was made in 2004[4;5] and has significantly changed models of the Earth's internal structure. In mid-2005 we had the first working version of USPEX. By September 2010, when USPEX was publicly released, the user community numbered nearly 200, over 800 users in May 2012, over 2100 in December 2014, and over 4500 in December 2018.

The popularity of USPEX is due to its extremely high efficiency and reliability. This was shown in the First Blind Test for Inorganic Crystal Structure Prediction[6], where USPEX outperformed the other methods it was tested against (simulated annealing and random sampling). Random sampling (a technique pioneered for structure prediction by Freeman and Schmidt in 1993 and 1996, respectively, and since 2006 revived by Pickard[7] under the name AIRSS) is the simplest, but also the least successful and computationally the most expensive strategy. Even for small systems, such as GaAs with 8 atoms/cell, these

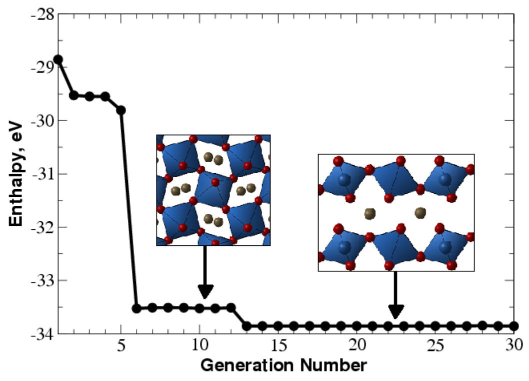


Figure 2: **Prediction of the crystal structure of $MgSiO_3$ at 120 GPa (20 atoms/cell).** Enthalpy of the best structure as a function of generation is shown. Between the 6th and 12th generations the best structure is perovskite, but at the 13th generation the global minimum (post-perovskite) is found. This simulation was performed in 2005 using one of the first versions of USPEX combined with *ab initio* calculations. It used no experimental information and illustrates that USPEX can find both the stable and low-energy metastable structures in a single simulation. Each generation contains 30 structures. This figure illustrates the slowest of ∼10 calculations performed by the very first version of USPEX — and even that was pretty fast!

advantages are large (random sampling requires on average 500 structure relaxations to find the ground state in this case, while USPEX finds it after only $\sim$ 30 relaxations! (Fig. 3)). Due to the exponential scaling of the complexity of structure search (eq. 1), the advantages of USPEX increase exponentially with system size. For instance, 2 out of 3 structures of SiH_4 predicted by random sampling to be stable[7], turned out to be unstable[8]; and similarly random sampling predictions were shown[9] to be incorrect for nitrogen[10] and for SnH_4 (compare predictions[11] of USPEX and of random sampling[12]).

For larger systems, random sampling tends to produce almost exclusively disordered structures with nearly identical energies, which decreases the success rate to practically zero, as shown in the example of $MgSiO_3$ post-perovskite with 40 atoms/supercell — random sampling fails to find the correct structure even after 120,000 relaxations, whereas USPEX finds it after several hundred relaxations (Fig. 4).

Random sampling runs can easily be performed with USPEX — but we see this useful mostly for testing. Likewise, the Particle Swarm Optimization (PSO) algorithm for cluster and crystal structure prediction (developed by A.I. Boldyrev and re-implemented by Wang, Lu, Zhu and Ma) has been revamped and implemented on the basis of USPEX with minor programming work as a corrected PSO (corPSO) algorithm, which outperforms previous versions of PSO. Still, any version of PSO is rather weak and we see the PSO approach suitable mostly for testing purposes, if anyone wants to try. A very powerful new method, complementary to our evolutionary algorithm, is evolutionary metadynamics[13], a hybrid of Martoňák's metadynamics and the Oganov-Glass evolutionary approach. This method is powerful for global optimization and for harvesting low-energy metastable structures, and even for finding possible phase transition pathways. For detailed investiga-

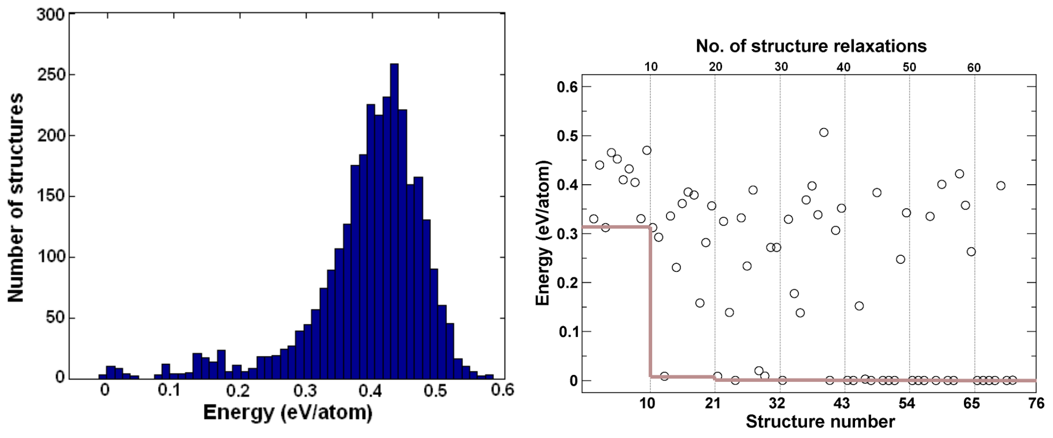


Figure 3: **Structure prediction for GaAs.** a) Energy distribution for relaxed random structures, b) progress of an evolutionary simulation (thin vertical lines show generations of structures, and the grey line shows the lowest energy as a function of generation). All energies are relative to the ground-state structure. The evolutionary simulation used 10 structures per generation. In addition, the lowest-energy structure of the previous generation survived into the next generation.

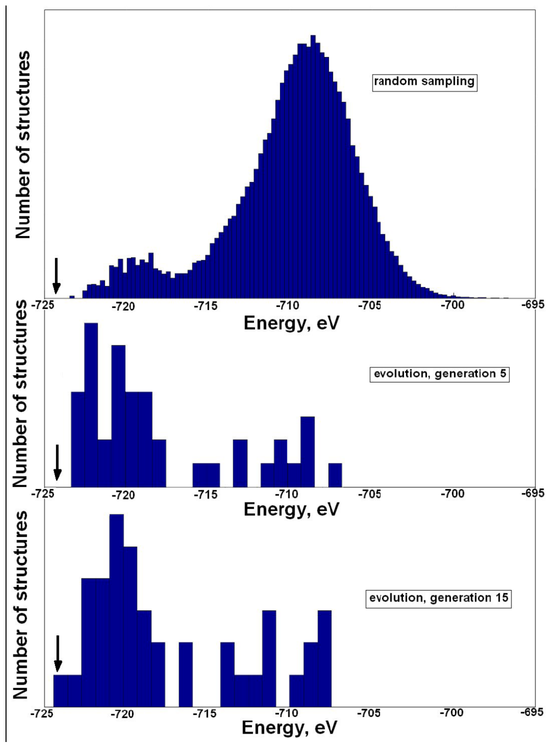


Figure 4: **Sampling of the energy surface: comparison of random sampling and USPEX for a 40-atom cell of $MgSiO_3$ with cell parameters of post-perovskite.** Energies of locally optimized structures are shown. For random sampling, 1.2×10^5 structures were generated (none of which corresponded to the ground state). For USPEX search, each generation included 40 structures and the ground-state structure was found within 15 generations. The energy of the ground-state structure is indicated by the arrow. This picture shows that "learning" incorporated in evolutionary search drives the simulation towards lower-energy structures.

tions of phase transition mechanisms, additional methods are implemented: variable-cell NEB method[14] and transition path method[15] in the version[16].

1.2 Features of USPEX

- Prediction of the stable and metastable structures knowing only the chemical composition. Simultaneous searches for stable compositions and structures are also possible.
- Incorporation of partial structural information is possible:
  - constraining search to fixed experimental cell parameters, or fixed cell shape, or fixed cell volume (Subsection 4.6);
  - starting structure search from known or hypothetical structures (Subsection 8.5);
  - assembling crystal structures from predefined molecules, including flexible molecules (Subsection 5.3).
- Efficient constraint techniques, which eliminate unphysical and redundant regions of the search space. Cell reduction technique (Oganov & Glass, 2008).
- Niching using fingerprint functions (Oganov & Valle, 2009; Lyakhov, Oganov, Valle, 2010). Subsection 4.9 for details.
- Initialization using fully random approach, or using space groups and cell splitting techniques (Lyakhov, Oganov, Valle, 2010). Use of powerful topological structure generator (Bushlanov, Blatov, Oganov, 2018).
- On-the-flight analysis of results — determination of space groups (and output in CIF-format) (Subsection 4.11), calculation of the hardness, order parameters, *etc.*
- Prediction of the structure of nanoparticles and surface reconstructions. See Section 5.4 for details.
- Restart functionality, enabling calculations to be continued from any point along the evolutionary trajectory (Subsection 4.7).
- Powerful visualization and analysis techniques implemented in the STM4 code (by M. Valle), fully interfaced with USPEX (Subsection 8.1).
- USPEX is interfaced with VASP, SIESTA, GULP, LAMMPS, DMACRYS, CP2K, Quantum Espresso, FHI-aims, ATK, CASTEP, Tinker, MOPAC codes. See full list of supported codes in Subsection 2.5.
- Submission of jobs from local workstation to remote clusters and supercomputers is possible. See Section 8.8 for details.

- Options for structure prediction using the USPEX algorithm (default), random sampling, corrected particle swarm optimization (Subsection 5.8), evolutionary metadynamics (Subsection 5.7), minima hopping-like algorithm. Capabilities to predict phase transition mechanisms using evolutionary metadynamics, variable-cell NEB method (Subsection 6.2), and TPS method (Subsection 6.3). USPEX also has a quick geometric mapping algorithm to predict likely mechanism of a phase transition. (Subsection 6.1)
- Options to optimize physical properties other than energy — *e.g.*, hardness (Lyakhov & Oganov, 2011), density (Zhu et al., 2011), band gap and dielectric constant (Zeng et al., 2014), and many other properties.
- Starting from version 9.4.1, USPEX has a Python-based runner of the code (`USPEX` Python module), providing a number of useful command line options.

1.3 Key USPEX papers

1. Oganov A.R., Glass C.W. (2006). Crystal structure prediction using evolutionary algorithms: principles and applications. *J. Chem. Phys.*, **124**, 244704.
2. Oganov A.R., Stokes H., Valle M. (2011). How evolutionary crystal structure prediction works — and why. *Acc. Chem. Res.*, **44**, 227–237.
3. Lyakhov A.O., Oganov A.R., Stokes H., Zhu Q. (2013). New developments in evolutionary structure prediction algorithm USPEX. *Comp. Phys. Comm.*, **184**, 1172–1182.
4. Zhu Q., Oganov A.R., Glass C.W., Stokes H. (2012). Constrained evolutionary algorithm for structure prediction of molecular crystals: methodology and applications. *Acta Cryst. B*, **68**, 215–226.
5. Zhu Q., Li L., Oganov A.R., Allen P.B. (2013). Evolutionary method for predicting surface reconstructions with variable stoichiometry. *Phys. Rev. B*, **87**, 195317.
6. Zhu, Q., Sharma V., Oganov A.R., Ramprasad R. (2014). Predicting polymeric crystal structures by evolutionary algorithms. *J. Chem. Phys.*, **141**, 154102.
7. Lyakhov A.O., Oganov A.R., Valle M. (2010). Crystal structure prediction using evolutionary approach. In: Modern methods of crystal structure prediction (ed: A.R. Oganov), Berlin: Wiley-VCH.
8. Oganov A.R., Ma Y., Lyakhov A.O., Valle M., Gatti C. (2010). Evolutionary crystal structure prediction as a method for the discovery of minerals and materials. *Rev. Mineral. Geochem.*, **71**, 271–298.

9. Duan, D., Liu, Y., Tian, F., Li, D., Huang, X., Zhao, Z., Yu, H., Liu, B., Tian, W., Cui, T. (2014). Pressure-induced metallization of dense $(H_2S)_2H_2$ with high-T_c superconductivity. *Sci. Rep.*, **4**, 6968.

10. Bushlanov P.V., Blatov V.A., Oganov A.R. (2019). Topology-based crystal structure generator. Comp. Phys. Comm., DOI: 10.1016/j.cpc.2018.09.016.

11. Lepeshkin S.V., Baturin V.S., Uspenskii Yu.A., Oganov A.R. (2019). Method for simultaneous prediction of atomic structure of nanoclusters in a wide area of compositions. J. Phys. Chem. Lett. 10, 102-106.

12. Allahyari Z., Oganov A.R. (2018). Multi-objective optimization as a tool for materials design. In:.Handbook of Materials Modeling (ed. W. Andreoni, S. Yip). Volume 2 Applications: Current and Emerging Materials. Springer Verlag.

1.4 Version history

v.1 — Evolutionary algorithm without local optimization. Real-space representation, interface with VASP. Experimental version. October 2004.

v.2 — CMA-ES implementation (CMA-ES is a powerful global optimization method developed by N. Hansen). Experimental version. January 2005.

v.3 — Evolutionary algorithm with local optimization.

v.3.1 — Working versions, sequential. Major basic developments.

3.1.4-3.1.5 — First production version. Based largely on heredity with slice-shifting and with minimum-parent contribution (hard-coded to be 0.25). May 2005.

3.1.8 — Adaptive k-point grids. 15/10/2005.

3.1.11 — Restart from arbitrary generation. Experimental version. 04/11/2005.

3.1.12 — Production version based on v.3.1.11, variable slice-shift mutation. 11/11/2005.

3.1.13 — Adaptive scaling volume. 29/11/2005.

3.1.14 — Basic seed technique. 29/11/2005 (debugged 6/12/2005).

v.3.2 — Massively parallel version.

v.4 — Unified parallel/sequential version.

4.1.1 — Lattice mutation. 20/12/2005 (debugged 10/01/2006).

4.2.1 — Interfaced with SIESTA. Initial population size allowed to differ from the running population size. 24/01/2006 (debugged 20/04/2006).

4.2.3 — Relaxation of best structures made optional. Version with fully debugged massive parallelism. 25/04/2006.

4.4.1 — Interfaced with GULP. 08/05/2006.

v.5 — Completely rewritten and debugged version, clear modular structure of the code.

5.1.1 — Atom-specific permutation, code interoperability, on-the-fly reading of parameters from `INPUT_EA.txt`. 20/12/2006.

5.2.1 — SIESTA-interface for Z-matrix, rotational mutation operator. 01/03/2007.

v.6 — Production version.

6.1.3 — To efficiently fulfill hard constraints for large systems, an optimizer was implemented within USPEX. 07/06/2007.

6.2 — Development version.

6.3.1–6.3.2 — Introduced angular constraints for cell diagonals. Completely rewritten remote submission. Improved input format. Further extended standard tests. 07/12/2007.

6.3.3 — X-com grid interface (with participation of S. Tikhonov and S. Sobolev). 05/03/2008.

6.4.1 — Fingerprint functions for niching. 07/04/2008.

6.4.4 — Space group recognition. Fast fingerprints (from tables). 05/05/2008.

6.5.1 — Split-cell method for large systems. Easy remote submission. Variable number of best structures (clustering). 16/07/2008.

6.6.1 — A very robust version — improved fingerprint and split-cell implementations. 13/08/2008.

6.6.3 — Heredity with multiple parents implemented. 01/10/2008.

6.6.4 — Added a threshold for parents participating in heredity (niching). 03/10/2008.

6.6.6 — First implementation of multicomponent fingerprints. 04/12/2008.

6.6.7, 6.7.1 and 6.7.2 — Implemented quasi-entropy to measure the diversity of the population. 10/12/2008.

v.7 — Production version, written to include variable composition.

7.1.1–7.1.7 — Series of improved versions. Version 7.1.7 has been distributed to ~200 users. Variable composition partly coded, most known bugs fixed, improved tricks based on energy landscapes. Improved cell splitting, implemented pseudo-subcells. Implemented multicomponent fingerprints (much more sensitive to the structure than one-component fingerprints). 28/04/2009 (version finalized 28/05/2009).

7.2.5 — First fully functional version of the variable-composition method. Introduced transmutation operator and compositional entropy. 06/09/2009.

7.2.7 — Thoroughly debugged, improved restart capabilities, improved seeding, introduced perturbations within structure relaxation. 25/09/2009, further improved in versions 7.2.8/9.

7.3.0 — Full fingerprint support in the variable-composition code, including niching. "Fair" algorithm for producing the first generation of compositions. 22/10/2009.

7.4.1 — Introduced coordinate mutation based on local order[17]. Heredity and transmutation are also biased by local order. Introduced computation of the hardness and new types of optimization by hardness and density. 04/01/2010.

7.4.2 — Implementation of multiple-parents heredity biased by local order. 15/01/2010.

7.4.3 — Implementation of new types of optimization (to maximize structural order and diversity of the population). Implemented antiseeds, eliminated parameters `volTimeConst`, `volBestHowMany`.

24/01/2010.

v.8 — Production version, written to include new types of optimization.

8.1.1–8.2.8 — Development versions. Local order and coordinate mutation operator. Softmutation operator. Calculation and optimization of the hardness. Optimization of the dielectric susceptibility. Prediction of the structure of nanoparticles and surfaces. Implementation of point groups. Greatly improved overall performance. Option to perform PSO simulations (not recommended for applications, due to PSO's inferior efficiency — so use only for testing purposes). Parameter `goodBonds` transformed into a matrix and used for building nanoparticles. 22/09/2010.

8.3.1 — Optimization of dielectric constants, cleaned-up input. 08/10/2010.

8.3.2 — For clusters, introduced a check on connectivity (extremely useful), `dynamicalBestHM`=2 option improved, as well as mechanism for producing purely softmutated generations. Improved fingerprints for clusters. Interface to Quantum Espresso and CP2K codes. 11/10/2010.

8.4 — Improved antiseed functionalities and several improvements for nanoparticles. Development branches for surface reconstructions, pseudo-metadynamics, molecular crystals.

8.5.0 — Initialization of the first random generation using the space group code of H. Stokes added. New formulation of metadynamics implemented and finalized, for now in a separate code. Several debugs for varcomp, antiseeds, nanoparticles, computation of hardness. 18/03/2011.

8.5.1 — Space group initialization implemented for cases of fixed unit cell, variable composition, and subcells. 20/04/2011.

8.6.0 — Added space group determination program from H. Stokes. Merger with the updated code for molecular crystals (including space group initialization). Fixed a bug for SIESTA (thanks to D. Skachkov). 06/05/2011.

8.6.1–8.7.2 — Improved symmetric initialization for the case of a fixed cell. Implemented optimization of dielectric constants (using GULP and VASP), band gap (using VASP), and DOS at the Fermi level (VASP). Graphical output enabled. Improved softmutation (by using better criteria for mode and directional degeneracies) and heredity (by using energy-order correlation coefficient and cosine formula for the number of trial slabs) operators. Most variables now have default values, which enables the use of very short input files. Shortened and improved the format of log-files. 13/11/2011.

8.7.5 — Graphical output now includes many extra figures. Added utility to extract all structures close to convex hull for easier post-processing. 21/03/2012.

v.9 — Production version, made more user-friendly and written to include new types of functionality and to set the new standard in the field.

9.0.0 — Evolutionary metadynamics and VCNEB codes added to USPEX package, added tensor version of metadynamics, added additional figures and post-processing tools, cleaned the code output. A few parameters removed from the input. Improved softmutation. April 2012.

9.1.0 — Release version. Cleaned up, documented. The user community is >800 people. Released 28/05/2012.

9.2.0 — Working GEM. Constant development of the GEM code. Space group determination tolerance is now an input parameter. Improved default for number of permutations. July-August 2012.

9.2.1–9.2.3 — Improved GEM, more diverse populations and supercell sizes, improved mode selection. September-October 2012.

9.2.4–9.2.6 — (9.2.4 is a release version). Intelligent defaults for most input parameters. Improved symmetric initialization for clusters. Order-enhanced heredity for nanoparticles. New parameter to tune the tolerance for the space group determination. New property (quasientropy) can be optimized. Fully integrated VCNEB code. November-December 2012.

9.2.7. — Release version. Enabled optimization of order for alloys, without structure relaxation (for easy creation of quasirandom structures, based on the more general definition than the so-called "special quasi-random structures"). Symmetry generation was improved (particularly important for fixed-cell calculations). For fixed-cell calculations, one can now specify the cell parameters, not only in the form of a 3×3 matrix, but also as a row of six values (three lengths in Angstroms and three angles in degrees). For the maximum number of permutation swaps (parameter `howManySwaps`), we have introduced an intelligent default. Added new tests, and cleaned and reran the old ones. Added interface to CASTEP (thanks to Z. Raza, X. Dong and AL). User community 1160 people. 30/12/2012.

9.3.0–9.3.3 — Fixed a bug in generation of random symmetric structures (this bug appeared in 9.2.7). Significantly simplified input and output. Created file `OUTPUT.txt` with the most important information. Enabled split-cell trick for molecular crystals. Improved variable-composition calculations by allowing one to specify initial compositions. Added interface to CASTEP and LAMMPS. Added new test cases. 20/03/2013.

9.3.4 — Release version, cleaned up. 25/03/2013.

9.3.5 — Added code for prediction of 2D-crystals. 19/04/2013.

9.3.6 — Incorporated plane groups for 2D-crystals. 29/04/2013.

9.3.8 — Incorporated plane groups for 1D-polymer crystals, improved variables of stoichiometry for surfaces. 19/06/2013.

9.3.9 — Released version. Significantly improved version, improved user-friendliness, new functionalities (2D-crystals, GEM) made more robust, improvements in the variable-composition algorithm (and enabled support for single-block calculations, *i.e.* fixed-composition searches with variable number of atoms in the cell), fully functional surface calculations, new optimization types (can optimize band gaps, dielectric constants, and newly invented figure of merit of dielectric materials). Interfaces with LAMMPS and ATK are documented in new test cases. Continuously updated with minor debugs (last debug 10/02/2014). 19/07/2013.

9.4.1 — A major upgrade, greatly improved user-friendliness (automatic estimate of volumes and of percentages of variation operators for each case), new functionalities (optimization of elastic properties and Chen's model of hardness, prediction of polymeric structures, anti-compositions, automatic analysis of statistics, improved seed technique), first release of GEM (generalized evolutionary metadynamics), provided a set of real-life examples of USPEX calculations, test cases, documentation. More than 2100 users. Released 30/12/2014.

9.4.2 — Release version, compatible with Octave 3.4. Convex hull code rewritten. Interface with MOPAC implemented. Default values for `goodBonds`, `valences`, `IonDistances` enabled. More robust for ternary, quaternary, and more complex variable-composition searches. Robust TPS implementation (only for developers, will be available for users soon). More than 2200 users. Released 21/03/2015.

9.4.3 — Release version. It includes fixing a number of bugs (which should slightly speed up performance), interface with MOPAC, improved documentation. Released 10/08/2015.

9.4.4 — Release version. It includes fix for space group determination and other problems reported by users, improved documentation and examples, full Octave 3.4 compatibility and partial Octave 3.6/3.8/4.0 support. This version should be nearly bug-free and is a milestone towards a very major upgrade, which will be made available in version 10. Released 05/10/2015.

10.1 - Experimental version, released as a virtual machine on 01/08/2018.

10.2 - Release version. Distributed as a compiled code, so users no longer need to have Matlab. Has no known bugs and contains a very large number of new features and improvements. Random topological structure generator and automatic parameter control greatly speed up calculations. Prediction of (collinear) magnetic materials is enabled. Many new types of fitness implemented: magnetization, birefringence, thermoelectric figure of merit ZT, fracture toughness. Fitness can be minimized or maximized, and we can input mathematical expressions as fitness. Pareto optimization of several fitnesses is enabled. USPEX has been interfaced with Gaussian, MOPAC, DFTB, ORCA, FHI-aims, ABINIT. Symmetry determination is now done using SPGLIB. Symmetrization can also be optionally performed during calculation of physical properties, making it cheaper and more robust. 80 layer symmetry groups are utilized for generating initial population of 2D-structures. Variable-composition prediction of 2D-crystals is enabled. For surface structure prediction, we have enabled all possible surface supercells and output the surface phase diagram. Variable-cell NEB method has been greatly improved (made a few times faster). Utility pmpaths of V. Stevanovic has been added to predict the likeliest phase transition mechanisms (which can then be directly input into the VCNEB code). Released 19/01/2019.

2 Getting started

2.1 How to obtain USPEX

USPEX can be downloaded at:

http://uspex-team.org

In the download page you will need to register and soon thereafter will receive a password for downloading USPEX. There are separate packages for USPEX itself, manual files and files for its installation.

2.2 Necessary citations

Whenever using USPEX, in all publications and reports you must cite the original papers, for example, in the following way:

> "Crystal structure prediction was performed using the USPEX code[2;13;18], based on an evolutionary algorithm developed by Oganov, Glass, Lyakhov and Zhu and featuring local optimization, real-space representation and flexible physically motivated variation operators".

Consult the `OUTPUT.txt` file for some of the most important references.

2.3 Bug reports

Like any large code, USPEX may have bugs. If you see strange behavior in your simulations, please report it to us in USPEX Google group at:

https://groups.google.com/forum/#!forum/uspex

Please describe your problem in details and attach `INPUT.txt`, `OUTPUT.txt`, `log` and other related files when you report a problem. You can also send the questions or problem descriptions to (bugreport@uspex-team.org)).

2.4 On which machines USPEX can be run

USPEX can be used on any unix-based platform — all you need is one CPU under Linux, Unix or Mac OS X with installed python and — using its special remote submission mechanism, USPEX will be able to connect to any remote machine and use it for calculations.

2.5 Codes that can work with USPEX

Trial structures generated by USPEX are relaxed and then evaluated by an external code interfaced with USPEX. Based on the obtained ranking of relaxed structures, USPEX generates new structures — which are again relaxed and ranked. Our philosophy is to use existing well-established *ab initio* (or classical forcefield) codes for structure relaxation and energy calculations. Currently, USPEX is interfaced with:

- VASP — `https://www.vasp.at/`
- SIESTA — `http://departments.icmab.es/leem/siesta/`
- GULP — `http://nanochemistry.curtin.edu.au/gulp/`
- LAMMPS — `http://lammps.sandia.gov/`
- DMACRYS — `http://www.chem.ucl.ac.uk/basictechorg/dmacrys/index.html`
- CP2K — `http://www.cp2k.org/`
- Quantum Espresso — `http://www.quantum-espresso.org/`
- FHI-aims — `https://aimsclub.fhi-berlin.mpg.de/`
- ATK — `http://quantumwise.com/`
- CASTEP — `http://www.castep.org/`
- Tinker — `http://dasher.wustl.edu/tinker/`
- MOPAC — `http://openmopac.net/`
- Gaussian — `http://www.gaussian.com/index.htm`
- ORCA — `https://orcaforum.cec.mpg.de`
- BoltzTraP — `http://www.imc.tuwien.ac.at/forschungsbereich_theoretische_chemie/forschungsgruppen/prof_dr_gkh_madsen_theoretical_materials_chemistry/boltztrap/`
- DFTB — `http://www.dftb-plus.info`

We chose these codes based on 1) their efficiency for structure relaxation; 2) robustness; and 3) popularity. Of course, there are other codes that can satisfy these criteria, and in the future we can interface USPEX to them.

2.6 How to install USPEX

USPEX v.10.2 is compiled using Matlab compiler in order to be used as a standalone code. Thus, there is NO need to install Matlab for using this version of the code. Instead, for running USPEX v.10.2, Matlab Runtime software (MCR-R2016b) must be installed on the target machine. The USPEX package contains proper version of MCR for running USPEX and the installation of both MCR and USPEX would be done together.

After you download the archive with USPEX, you need to unpack it and run the following command to install USPEX to a user's or system-wide location:

```
sh install.sh
```

The installer does not require root privileges, unless you want to install codes in the root partition. For installing USPEX v10.2 together with MCR-R2016b one needs four simple steps:

- Choose the method for installation (graphical or terminal installation)
- Accept terms and conditions of both USPEX and MCR user licenses.
- Select the installation directory of USPEX and MCR (If the installation directory was chosen to be in the root partition, the user needs to have root privilege).
- Setting up the information about environmental variables (it will be done automatically in terminal installation).

For the last step, if the terminal installation is selected, information about environmental variables will be added automatically to the `.bashrc` (bash shell system) or `.cshrc` (C shell system). For graphical installation, you will be provided with information about environmental variables during the installation process, which must be set by the user, to make USPEX available in the system. For example:

```
For Bash shell system, add these lines in ~/.bashrc or ~/.profile or /etc/profile:
     export PATH=/home/user/bin/USPEX/application/archive/:$PATH
     export USPEXPATH=/home/user/bin/USPEX/application/archive/src
     export MCRROOT=/usr/local/MATLAB/MATLAB_Runtime

For C shell system, add these lines in ~/.cshrc or ~/.profile or /etc/profile:
     setenv PATH "/home/user/bin/USPEX/application/archive :$PATH"
     setenv USPEXPATH "/home/user/bin/USPEX/application/archive/src"
     setenv MCRROOT /usr/local/MATLAB/MATLAB_Runtime
```

*** Please note that MCRROOT is the path before the folder v91.

*** It is NOT recommended to install the code in the root partition.

2.7 How to run USPEX

To run USPEX, you need to have **Python** and to have the executable of the external code on the compute nodes that you use for relaxing structures and computing their energies (see Subsection 2.5 for a list of supported codes). To set up your calculation, find an example (see Appendix 9.1 for a list of examples) similar to what you want to do, and start by editing `INPUT.txt`. The variables of this crucial file are described in Section 4 below. Then, gather the files needed for the external code performing structure relaxation in the `Specific/` folder — the executable (*e.g.*, `vasp`), and such files `INCAR_1`, `INCAR_2`, ..., `INCAR_N`, and `POTCAR_A`, `POTCAR_B`, ..., where `A`, `B`, ... are the symbols of the chemical elements of your compound.

Another important thing to start your calculation are scripts for running (localy or remotely) the external code, depending on whether you submit *ab initio* calculations on the same machine where you run USPEX, or if you send your jobs to a remote supercomputer. See the keyword `whichCluster` and Section 8.8 of this Manual.

To run USPEX just type:

```
    USPEX -r
```

File log will contain information on the progress of the simulation and, if any, errors (send these to us, if you would like to report a bug). File OUTPUT.txt will contain details of the calculation and an analysis of each generation.

For the USPEX runner, we have a number of user-friendly options:

-v, --version: show program's version number and exit

-h, --help: show help message and exit

-e, --example: display details about any supplied example. If no value or 'all' value is specified, all examples will be shown

-c NUM, --copy=NUM: copy the INPUT.txt file and Specific Folder of ExampleXX.

-r, --run: run USPEX calculation

--clean: clean calculation folder

--list: show id(s) existing calculation(s)

When running USPEX in the massively parallel mode, the user needs to do minimal work to configure files to the user's computers.

There are two modes for job submission — (1) local submission and (2) remote submission, depending on whether you submit *ab initio* calculations on the same machine where you run USPEX and MATLAB, or if you send your jobs to a remote supercomputer. See the keyword `whichCluster` and Section 8.8 of this Manual.

2.8 Running USPEX examples: a mini-tutorial

Once you have downloaded the USPEX package and installed it, you can run the first USPEX example. The description of the examples is listed in Appendix 9.1. The required external codes to run the examples (except EX13) are shown below:

- GULP: EX02, EX03, EX08, EX12, EX15 (VCNEB), EX16, EX18, EX21(META), EX22(Generalized META), EX23
- VASP: EX01, EX07, EX09, EX14 (META), EX17, EX19, EX24, EX29, EX30, EX31
- LAMMPS: EX04, EX26 (TPS)
- ATK: EX05
- CASTEP: EX06
- DMACRYS: EX10
- Tinker: EX11
- MOPAC: EX20
- DFTB: EX25
- FHIaims: EX27, EX28

Now, let us start our first USPEX experience:

2.8.1 Test the USPEX python runner

To get the version information, you can use the following command:

```
>> USPEX -v
```

You should get something like this:

```
  USPEX Version 10.2 (19/01/2019)
```

2.8.2 Run the EX13-3D_special_quasirandom_structure_TiCoO2

EX13 does not require any external code, you can run it to get familiar with the USPEX running procedure. The calculation would take ∼30 minutes. To start the calculation, first create a test folder, copy the example files and then run the calculation through the USPEX Python runner, with the following commands:

```
>> mkdir EX13
>> cd EX13
>> USPEX -c 13
>> USPEX -r
```

Meanwhile you have some time to get more details about example EX13. In EX13, the structural order is optimized (minimized) with the evolutionary algorithm to yield a generalized special quasirandom structure (gSQS). Therefore in `INPUT.txt` the following is set:

```
USPEX : calculationMethod
% optType
-4
% EndOptType
```

and these parameters are used below:

```
300   : calculationType

% atomType
Co Ti O
% EndAtomType

% numSpecies
16 16 64
% EndNumSpecies
```

which specifies that we deal with the $Co_{16}Ti_{16}O_{64}$ system.

Since you do not need to use any external code, you can simply set

```
% abinitioCode
0
% ENDabinit
```

The `Seeds/POSCARS` file contains the initial $Ti_{16}Co_{16}O_{64}$ structure.

When you find the `USPEX_IS_DONE` file, congratulations, you have successfully finished your first USPEX example. Next, you can run calculations which require external codes.

2.8.3 Run an example using external code

In this step, we suggest to run examples interfaced with GULP or VASP, starting from EX02 or EX01. Use USPEX runner to get the example information of EX02, create a separate folder and copy the files, with commands:

```
>> mkdir EX02
>> cd EX02
>> USPEX -c 2
```

Since in example EX02 GULP code is used, set:

```
% abinitioCode
3 3 3 3
% ENDabinit
```

To run a serial job without a job batch system, you should change the following parameters in the `INPUT.txt` below:

```
1     : whichCluster
1     : numParallelCalcs
```

In the example `INPUT.txt`, you will see `whichCluster`=QSH, where QSH is name of our group's cluster name. User can specify their own cluster: for the details, please see Section 8.8.

```
% commandExecutable
gulp < input > output
% EndExecutable
```

But make sure, that this command also works on your machine. If you want to run EX01 with a parallel version of VASP, you should set something like:

```
% abinitioCode
1 1 1 1
% ENDabinit

% commandExecutable
mpirun -np 8 vasp
% EndExecutable
```

With wrong `commandExecutable` settings, you will not able to run your calculation. Finally, do not forget to suitably modify `submitJob_remote.py` and `checkStatus_remote.py` files for computer/supercomputer.

When everything is set correctly, we can run the calculation through USPEX runner using the command:

```
>> USPEX -r
```

2.8.4 Checking the results

After starting the command, you can check the output files in results1/ folder.

Now, you have a basic experience of using USPEX to run simple calculations. Please read the following sections of this manual to get more insight into USPEX. When analyzing results, it is essential that you visualize the structures (for visualization, see section 8.1).

3 Overview of input and output files

Input/output files depend on the external code used for structure relaxation.

An important technical element of our philosophy is the multi-stage strategy for structure relaxation. Final structures and energies must be high-quality, in order to correctly drive evolution. Most of the newly generated structures are far from local minimum and their high-quality relaxation is extremely expensive. This cost can be offset if the first stages of relaxation are done with cruder computational conditions — only at the last stages is there a need for high-quality calculations. The first stages of structure relaxation can be performed with cheaper approaches or lower computational conditions (basis set, k-points sampling, pseudopotentials) or level of approximation (forcefields *vs.* LDA *vs.* GGA) and even different structure relaxation code (see Subsection 2.5 for a list of supported codes) during structure relaxation of each candidate structure. We strongly suggest you initially optimize the cell shape and atomic positions at constant unit cell volume, and only then perform full optimization of all structural variables. While optimizing at constant volume, you do not need to worry about Pulay stresses in plane-wave calculations — it is OK to use a small basis set; however, for variable-cell relaxation you will need a high-quality basis set. For structure relaxation, you can often get away with a small set of k-points — but don't forget to sufficiently increase this at the last stage(s) of structure relaxation, to get accurate energies. Use your (and our) wisdom, be a strategist, and remember that poor relaxation can ruin your results.

3.1 Input files

Suppose that the directory where the calculations are performed is `~/StructurePrediction`. This directory will contain:

- file `INPUT.txt`, thoroughly described in Section 4.
- Subdirectory `~/StructurePrediction/Specific/` with VASP, SIESTA or GULP (*etc.*) executables, and enumerated input files for structure relaxation — `INCAR_1`, `INCAR_2`, ..., and pseudopotentials.
- Subdirectory `~/StructurePrediction/Seeds` — contains files with seed structures and a list of compositions/anti-compositions. Seed structures should be in VASP5 POSCAR format and concatenated in a file called `POSCARS` or `POSCARS_gen` (`gen` is the generation number). The `compositions` and `Anti-compositions` files are used to control the compositions during variable-composition or single-block calculations.
- Subdirectory `~/StructurePrediction/AntiSeeds` — you may put here particular structures that you wish to penalize.

3.1.1 `Specific/` folder

Executables and enumerated input files for structure relaxation (using external codes, like VASP, SIESTA, GULP, ...) should be put in subdirectory ~`/StructurePrediction/Specific/`

- For VASP, put files `INCAR_1`, `INCAR_2`, ..., *etc.*, defining how relaxation and energy calculations will be performed at each stage of relaxation (we recommend at least 3 stages of relaxation), and the corresponding `POTCAR_*` files with pseudopotentials. *E.g.*, `INCAR_1` and `INCAR_2` perform very crude structure relaxation of both atomic positions and cell parameters, keeping the volume fixed, `INCAR_3` performs full structure relaxation under constant pressure with medium precision, `INCAR_4` performs very accurate calculations. Each higher-level structure relaxation starts from the results of a lower-level optimization and improves them. `POTCAR` files of all relevant elements should also be in `Specific/` folder, for instance `POTCAR_C`, `POTCAR_O`, *etc.*
- For SIESTA, you need the pseudopotentials files and input files `input_1.fdf`, `input_2.fdf`, ...
- For GULP, files `goptions_1`, `goptions_2`, ..., and `ginput_1`, `ginput_2`, ... must be present. The former specify what kind of optimization is performed, the latter specify the details (interatomic potentials, pressure, temperature, number of relaxation iterations, *etc.*).
- For DMACRYS, `fort.22` is the file for general control parameters. The classical force field is given by the file of `fit.pots`. File `cutoff` defines the maximum bond length of the intra-molecular bonds.
- For CASTEP, structural files are given by `cell_1`, `cell_2`, ..., while the computational parameters are given by `param_1`, `param_2`, ... The corresponding pseudopotential files must be present as well.
- For CP2K, files `cp2k_options_1`, `cp2k_options_2`, ..., must be present. All files should be normal CP2K input files with all parameters **except** atomic coordinates and cell parameters (these will be written by USPEX together with the finishing line "`&END FORCE_EVAL`"). The "`name of the project`" should always be USPEX, since the program reads the output from files `USPEX-1.cell` and `USPEX-pos-1.xyz`. We recommend performing relaxation at least in three steps (similarly to VASP) — first optimize only the atom positions with the lattice fixed, and then do a full relaxation.
- For Quantum Espresso, files `qEspresso_options_1`, `qEspresso_options_2`, ..., must be present. All files should be the normal QE input files with all parameters except atom coordinates, cell parameters and k-points (these will be written by USPEX at the end of the file). We recommend performing a multi-step relaxation.

For instance, `qEspresso_options_1` does a crude structure relaxation of atomic positions with fixed cell parameters, `qEspresso_options_2` does full structure relaxation under constant external pressure with medium precision; and `qEspresso_options_3` does very accurate calculations.

`INCAR_*` files in `Specific/` folder for VASP To use USPEX correctly, you should carefully edit the files in `Specific/` folder to control the structure relaxation in USPEX. We take example of VASP as an external code:

- Your final structures have to be well relaxed, and energies — precise. The point is that your energy ranking has to be correct (to check this, look at `E_series.pdf` file in the output).
- Your `POTCAR` files: To yield correct results, the cores of your pseudopotentials (or PAW potentials) should not overlap by more than 10–15%.
- To have accurate relaxation at low cost, use the multistage relaxation with at least three stages of relaxation for each structure, *i.e.* at least three `INCAR` files (`INCAR_1`, `INCAR_2`, `INCAR_3`, . . .). We usually set 4–5 stages of relaxation.
- Your initial structures will be usually very far from local minima, in such cases it helps to relax atoms and cell shape at constant volume first (`ISIF`=4 in `INCAR_1,2`), then do full relaxation (`ISIF`=3 in `INCAR_3,4`), and finish with a very accurate single-point calculation (`ISIF`=2 and `NSW`=0 in `INCAR_5`).

  Exceptions: when you do fixed-cell predictions, and also in evolutionary metadynamics (except full relaxation) you must have `ISIF`=2.
- When your volume does not change, you can use default plane wave cutoff. When you optimize cell voluem (`ISIF`=3), you must increase it by 30–40%, otherwise you get a large Pulay stress. Also your convergence criteria can be loose in the beginning, but have to be tight in the end: *e.g.*, `EDIFF`=1e-2 and `EDIFFG`=1e-1 in `INCAR_1`, gradually tightening to `EDIFF`=1e-4 and `EDIFFG`=1e-3 in `INCAR_4`. The maximum number of iterations (`NSW`) should be sufficiently large to enable good relaxation, but not too large to avoid wasting computer time on poor configurations. The larger your system, the larger `NSW` should be.
- Choosing an efficient relaxation algorithm can save a lot of time. In VASP, we recommend to start relaxation with conjugate gradients (`IBRION`=2 and `POTIM`=0.02) and when the structure is closer to local minimum, switch to `IBRION`=1 and `POTIM`=0.3.
- Even if you study an insulating system, many configurations that you will sample are going to be metallic, so to have well-converged results, you must use "metallic" treatment — which works both for metals and insulators. We recommend the Methfessel-Paxton smearing scheme (`ISMEAR`=1). For a clearly metallic system,

use ISMEAR=1 and SIGMA=0.1–0.2. For a clearly insulating system, we recommend ISMEAR=1 and SIGMA starting at 0.1 (INCAR_1) and decreasing to 0.05.

Here we provide an example of INCAR files for carbon with 16 atoms in the unit cell, with default ENCUT=400 eV in POTCAR:

```
INCAR_1:
    PREC=LOW
    EDIFF=1e-2
    EDIFFG=1e-1
    NSW=65
    ISIF=4
    IBRION=2
    POTIM=0.02
    ISMEAR=1
    SIGMA=0.10
```

```
INCAR_2:
    PREC=NORMAL
    EDIFF=1e-3
    EDIFFG=1e-2
    NSW=55
    ISIF=4
    IBRION=1
    POTIM=0.30
    ISMEAR=1
    SIGMA=0.08
```

```
INCAR_3:
    PREC=NORMAL
    EDIFF=1e-3
    EDIFFG=1e-2
    ENCUT=520.0
    NSW=65
    ISIF=3
    IBRION=2
    POTIM=0.02
    ISMEAR=1
    SIGMA=0.07
```

```
INCAR_4:
    PREC=NORMAL
    EDIFF=1e-4
    EDIFFG=1e-3
    ENCUT=600.0
    NSW=55
    ISIF=3
    IBRION=1
    POTIM=0.30
    ISMEAR=1
    SIGMA=0.06
```

```
INCAR_5:
    PREC=NORMAL
    EDIFF=1e-4
    EDIFFG=1e-3
    ENCUT=600.0
    NSW=0
    ISIF=2
    IBRION=2
    POTIM=0.02
    ISMEAR=1
    SIGMA=0.05
```

The philosophy of input files for evolutionary metadynamics (calculationMethod=META) is very similar to USPEX, except that we DO NOT change the cell shape during the META evolution. Therefore, we need to put ISIF=2 for all META steps. If the full relaxation mode is on, we can put ISIF=3 for the steps of full relaxation. Therefore, if we have the following set up:

```
% abinitioCode
1 1 1 (1 1)
% ENDabinit
```

the ISIF should be "2 2 2 3 3" for INCAR_1, . . . , INCAR_5 correspondingly.

Different from USPEX, VCNEB method doesn't need structure relaxation from the external codes, and itself makes use of the forces completed by the external code. Take

VASP `INCAR` files for example, we need to set `NSW`=0 to avoid the structure relaxation, but with `ISIF`=2 or 3 to extract the forces on the atoms, and the stress tensor in VASP. We also suggest to use `PREC`=Accurate to have a good estimation of the forces and stress for VCNEB. An example of `INCAR` file for VCNEB is presented below:

```
INCAR_1:
    PREC=Accurate
    EDIFF=1e-4
    EDIFFG=1e-3
    ENCUT=600.0
    NSW=0
    ISIF=2
    IBRION=2
    POTIM=0.02
    ISMEAR=1
    SIGMA=0.05
```

3.2 Output files

These are stored in the folder ∼`/StructurePrediction/results1`. If this is a new calculation, `results2`, `results3`, . . . (if the calculation has been restarted or run a few times), there will be a separate `results*` folder for each calculation. **Caution:** When looking at space groups in the file Individuals, keep in mind that USPEX often underdetermines space group symmetries, because of finite precision of structure relaxation and relatively tight space group determination tolerances. You should visualize the predicted structures. To get full space group, either increase symmetry tolerances (but this can be dangerous), or re-relax your structure with increased precision.

The subdirectory ∼`/StructurePrediction/results1` contains the following files:

- `OUTPUT.txt` — summarizes input variables, structures produced by USPEX, and their characteristics.
- `Parameters.txt` — this is a copy of the `INPUT.txt` file used in this calculation, for your reference.
- `Individuals` — gives details of all produced structures (energies, unit cell volumes, space groups, variation operators that were used to produce the structures, k-points mesh used to compute the structures' final energy, degrees of order, *etc.*). File `BESTIndividuals` gives this information for the best structures from each generation. Example of `Individuals` file:

```
Gen   ID    Origin   Composition     Enthalpy   Volume  Density    Fitness    KPOINTS  SYMM  Q_entr A_order S_order
                                       (eV)      (A^3)  (g/cm^3)
  1    1   Random    [    4   8 16 ] -655.062   201.062   4.700    -655.062 [ 1   1   1]    1   0.140  1.209  2.632
  1    2  Random    [   4  8 16 ] -650.378   206.675   4.572   -650.378 [ 1  1  1]   1   0.195  1.050  2.142
  1    3  Random    [   4  8 16 ] -646.184   203.354   4.647   -646.184 [ 1  1  1]   1   0.229  0.922  1.746
  1    4  Random    [   4  8 16 ] -649.459   198.097   4.770   -649.459 [ 1  1  1]   9   0.128  0.958  2.171
  1    5  Random    [   4  8 16 ] -648.352   202.711   4.662   -648.352 [ 1  1  1]   2   0.154  1.014  2.148
  1    6  Random    [   4  8 16 ] -643.161   206.442   4.577   -643.161 [ 1  1  1]   1   0.234  0.946  1.766
  1    7  Random    [   4  8 16 ] -647.678   207.119   4.562   -647.678 [ 1  1  1]   1   0.224  1.108  2.106
  1    8  Random    [   4  8 16 ] -644.482   203.844   4.636   -644.482 [ 1  1  1]   1   0.215  0.952  1.857
  1    9  Random    [   4  8 16 ] -647.287   204.762   4.615   -647.287 [ 1  1  1]  40   0.136  1.142  2.563
  1   10  Random    [   4  8 16 ] -649.459   198.097   4.770   -649.459 [ 1  1  1]   9   0.128  0.958  2.171
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

- `convex_hull` — only for variable-composition calculations, this file gives all thermodynamically stable compositions, and their enthalpies (per atom). Example:

```
---- generation  1 -------
  10   0     -8.5889
   0  14     -8.5893
  11   3     -8.7679
---- generation  2 -------
  10   0     -8.5889
   0  14     -8.5893
  11   3     -8.8204
---- generation  3 -------
  10   0     -8.5889
   0  14     -8.5893
  12   4     -8.9945
. . . . . . . . . . . . .
```

- `gatheredPOSCARS` — relaxed structures (in the VASP5 POSCAR format). Example:

```
EA1     9.346  8.002  2.688 90.000 90.000 90.000 Sym.group:    1
1.0000
    9.346156     0.000000     0.000000
    0.000000     8.002181     0.000000
    0.000000     0.000000     2.688367
   Mg   Al  O
   4    8   16
Direct
    0.487956     0.503856     0.516443
    0.777565     0.007329     0.016443
    0.987956     0.507329     0.016443
    0.277565     0.003856     0.516443
    0.016944     0.178753     0.016443
    0.019294     0.833730     0.516443
    0.746227     0.333730     0.516443
    0.748577     0.678753     0.016443
    0.516944     0.832431     0.516443
    0.519294     0.177455     0.016443
    0.246227     0.677455     0.016443
    0.248577     0.332431     0.516443
    0.416676     0.241774     0.516443
    0.559871     0.674713     0.016443
    0.205650     0.174713     0.016443
    0.348845     0.741774     0.516443
    0.613957     0.380343     0.016443
    0.804054     0.542164     0.516443
    0.113957     0.630842     0.516443
    0.304054     0.469021     0.016443
    0.848845     0.269411     0.016443
    0.705650     0.836472     0.516443
    0.059871     0.336472     0.516443
    0.916676     0.769411     0.016443
    0.651564     0.130842     0.516443
    0.461467     0.969021     0.016443
    0.151564     0.880343     0.016443
    0.961467     0.042164     0.516443
EA2     9.487  4.757  4.580 90.243 90.188 89.349 Sym.group:    1
1.0000
    9.486893     0.000000     0.000000
    0.054041     4.756769     0.000000
   -0.014991    -0.019246     4.579857
   Mg   Al  O
   4    8   16
Direct
    0.499837     0.633752     0.011361
    0.500082     0.131390     0.482012
    0.813573     0.257696     0.494520
    0.326111     0.625491     0.501746
    0.995267     0.254346     0.992293
    0.160822     0.689054     0.001270
    0.995907     0.760753     0.498354
    0.159742     0.192300     0.491958
    0.811206     0.761223     0.997857
    0.325692     0.125479     0.987935
    0.656355     0.695175     0.503322
    0.656596     0.199917     0.991605
    0.487990     0.763078     0.627771
    0.845518     0.645378     0.347890
    0.623474     0.895186     0.185946
    0.616379     0.395875     0.308861
    0.093745     0.991831     0.185467
    0.092669     0.494591     0.309957
    0.847697     0.118765     0.113434
    0.475636     0.251449     0.875207
    0.327510     0.787484     0.116764
    0.720411     0.975740     0.706398
    0.200804     0.880147     0.683027
    0.975416     0.612789     0.852917
    0.986131     0.108285     0.644081
    0.204805     0.364607     0.830780
    0.718464     0.496262     0.817031
    0.323904     0.257705     0.340590
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

- `BESTgatheredPOSCARS` — the same data for the best structure in each generation.
- `gatheredPOSCARS_unrelaxed` — gives all structures produced by USPEX before relaxation.
- `enthalpies_complete.dat` — gives the enthalpies for all structures in each stage of relaxation.
- `origin` — shows which structures originated from which parents and through which variation operators. Example:

```
 ID    Origin    Enthalpy   Parent-E   Parent-ID
   1   Random     -23.395   -23.395  [        0]
   2   Random     -23.228   -23.228  [        0]
   3   Random     -23.078   -23.078  [        0]
   4   Random     -23.195   -23.195  [        0]
   5   Random     -23.155   -23.155  [        0]
   6   Random     -22.970   -22.970  [        0]
   7   Random     -23.131   -23.131  [        0]
   8   Random     -23.017   -23.017  [        0]
   9   Random     -23.117   -23.117  [        0]
  10   Random     -23.195   -23.195  [        0]
. . . . . . . . . . . . . . . . . . . . . . . . .
```

- `gatheredPOSCARS_order` — gives the same information as `gatheredPOSCARS`, and in addition for each atom it gives the values of local order parameters (Ref.[17]). Example:

```
EA1     9.346  8.002  2.688 90.000 90.000 90.000 Sym.group:    1
1.0000
    9.346156     0.000000     0.000000
    0.000000     8.002181     0.000000
    0.000000     0.000000     2.688367
   Mg   Al  O
   4    8   16
Direct
    0.487956     0.503856     0.516443     1.1399
    0.777565     0.007329     0.016443     1.1399
    0.987956     0.507329     0.016443     1.1399
    0.277565     0.003856     0.516443     1.1399
    0.016944     0.178753     0.016443     1.1915
    0.019294     0.833730     0.516443     1.2474
    0.746227     0.333730     0.516443     1.2474
    0.748577     0.678753     0.016443     1.1915
    0.516944     0.832431     0.516443     1.1915
    0.519294     0.177455     0.016443     1.2474
    0.246227     0.677455     0.016443     1.2474
    0.248577     0.332431     0.516443     1.1915
    0.416676     0.241774     0.516443     1.2914
    0.559871     0.674713     0.016443     1.1408
    0.205650     0.174713     0.016443     1.1408
    0.348845     0.741774     0.516443     1.2914
    0.613957     0.380343     0.016443     1.2355
    0.804054     0.542164     0.516443     1.2161
    0.113957     0.630842     0.516443     1.2355
    0.304054     0.469021     0.016443     1.2161
    0.848845     0.269411     0.016443     1.2914
    0.705650     0.836472     0.516443     1.1408
    0.059871     0.336472     0.516443     1.1408
    0.916676     0.769411     0.016443     1.2914
    0.651564     0.130842     0.516443     1.2355
    0.461467     0.969021     0.016443     1.2161
    0.151564     0.880343     0.016443     1.2355
    0.961467     0.042164     0.516443     1.2161
EA2     9.487  4.757  4.580 90.243 90.188 89.349 Sym.group:    1
1.0000
    9.486893     0.000000     0.000000
    0.054041     4.756769     0.000000
   -0.014991    -0.019246     4.579857
   Mg   Al  O
   4    8   16
Direct
    0.499837     0.633752     0.011361     0.9368
    0.500082     0.131390     0.482012     1.0250
    0.813573     0.257696     0.494520     0.9805
    0.326111     0.625491     0.501746     0.9437
    0.995267     0.254346     0.992293     0.9241
    0.160822     0.689054     0.001270     1.1731
    0.995907     0.760753     0.498354     0.9696
    0.159742     0.192300     0.491958     1.2666
    0.811206     0.761223     0.997857     1.0215
    0.325692     0.125479     0.987935     1.0353
    0.656355     0.695175     0.503322     1.2291
    0.656596     0.199917     0.991605     1.2547
    0.487990     0.763078     0.627771     1.1199
    0.845518     0.645378     0.347890     0.9725
    0.623474     0.895186     0.185946     0.9990
    0.616379     0.395875     0.308861     1.0141
    0.093745     0.991831     0.185467     1.1451
    0.092669     0.494591     0.309957     1.1164
    0.847697     0.118765     0.113434     0.8821
    0.475636     0.251449     0.875207     1.0609
    0.327510     0.787484     0.116764     1.0451
    0.720411     0.975740     0.706398     0.9539
    0.200804     0.880147     0.683027     0.9398
    0.975416     0.612789     0.852917     1.1191
    0.986131     0.108285     0.644081     0.9803
    0.204805     0.364607     0.830780     1.0915
    0.718464     0.496262     0.817031     1.0663
    0.323904     0.257705     0.340590     1.1471
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

- `goodStructures_POSCARS` and `extended_convex_hull_POSCARS` (for fixed- and variable-composition calculations correspondingly) report all of the different structures in order of decreasing stability, starting from the most stable structure and ending with the least stable.

- `compositionStatistics` is a file containing statistics of the compositions in terms of which variation operators produced these compositions. Example:

```
      Comp/Ratio        Total     Random Heredity Mutation Seeds  COPEX  Best/Convex
 [  0.0000  1.0000  ]  30( 63)      21       8       1       0       0       33
        0      8         4(  4)      2       2       0       0       0        0
        0      9         2(  2)      2       0       0       0       0        0
        0     10         5(  5)      3       2       0       0       0        0
        0     11         2(  2)      0       1       1       0       0        0
        0     12         6( 35)      6       0       0       0       0       29
        0     13         2(  3)      1       1       0       0       0        1
        0     14         0(  0)      0       0       0       0       0        0
        0     15         2(  2)      2       0       0       0       0        0
        0     16         5(  8)      5       0       0       0       0        3
        0      3         1(  1)      0       1       0       0       0        0
        0      4         1(  1)      0       1       0       0       0        0
 [  0.1250  0.8750  ]  52( 52)      32       9      11       0       0        0
        1      7        20( 20)      4       8       8       0       0        0
        2     14        32( 32)     28       1       3       0       0        0
```

```
[  0.1111  0.8889  ]   25( 25)      9      14       2       0       0       0
       1     8         25( 25)      9      14       2       0       0       0
[  0.1000  0.9000  ]   16( 16)      9       5       2       0       0       0
       1     9         16( 16)      9       5       2       0       0       0
[  0.0909  0.9091  ]   11( 11)      5       4       2       0       0       0
       1    10         11( 11)      5       4       2       0       0       0
[  0.0833  0.9167  ]   17( 17)      8       2       7       0       0       0
       1    11         14( 14)      8       2       4       0       0       0
       2    22          3(  3)      0       0       3       0       0       0
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

- graphical files (`*.pdf`) — for rapid visual assessment of the results:
  - `Energy_vs_N.pdf` (`Fitness_vs_N.pdf`) — energy (fitness) as a function of structure number;
  - `Energy_vs_Volume.pdf` — energy as a function of volume;
  - `Variation-Operators.pdf` — energy of the child *vs.* parent(s); different operators are marked with different colors (this graph allows one to assess the performance of different variation operators) also show evolution of each operator's strength.
  - `E_series.pdf` — correlation between energies from relaxation steps i and $i+1$; helps to detect problems and improve structure relaxation.
  - For variable compositions there is an additional graph `extendedConvexHull.pdf`, which shows the enthalpy of formation as function of composition.
  - `compositionStatistics.pdf` — visualization of `compositionStatistics` file.

4 Input options. The INPUT.txt file

Typical INPUT.txt files are given in the Appendix 9.3. Below we describe the most important parameters of the input. Most of the parameters have reliable default values (this allows you to have extremely short input files!). Those options that have no default, and should always be specified. Please consult online utilities at http://http://uspex-team.org/online_utilities/ — these help to prepare the INPUT.txt file, molecular files, and analyze some of results. Section 7 of this Manual briefly discusses these utilities.

4.1 Type of calculation and system specification

▷ *variable* calculationMethod

Meaning: Specifies the method of calculation

Possible values (characters):

- USPEX — evolutionary algorithm for crystal structure prediction
- META — evolutionary metadynamics
- VCNEB — transition path determination using the variable-cell nudged elastic band method
- PSO — corrected PSO algorithm
- TPS — transition path sampling method
- MINHOP — minima hopping method
- COPEX — coevolutionary technique for reliable variable-composition ternary runs

Default: `USPEX`

Format:

```
USPEX : calculationMethod
```

▷ *variable* calculationType

Meaning: Specifies type of calculation, *i.e.*, whether the structure of a bulk crystal, nanoparticle, or surface is to be predicted. This variable consists of three indices: *dimensionality*, *molecularity* and *compositional variability*, and the spin option with character “s” or “S”:

- dimensionality:

"3" — bulk crystals

"2" — surfaces, "–2" — 2D-crystals

"1" — polymers

"0" — nanoparticles

- molecularity:

  "0" — non-molecular

  "1" — molecular calculations

- variability of chemical composition in the calculation:

  "0" — fixed composition

  "1" — variable composition

- magnetic calculation :

  "s" or "S" – enable the magnetic calculation

Default: `300`

Format:

```
301 : calculationType
```

Note: If `calculationType`=310, *i.e.*, a prediction for a molecular crystal is to be performed, then USPEX expects you to provide files `MOL_1`, `MOL_2`, . . . with molecular geometries for all types of molecules, and these molecules will be placed in the newly generated structures as whole objects. Available options: 300 (s300), 301 (s301), 310, 311, 000 (s000), 200 (s200), 201 (s201), –200 (–s200).

▷ *variable* `optType`

Meaning: This keyblock specifies the property (or properties) that you want to optimize. Default is minimization for enthalpy (and finite-temperature free energy) and volume, and maximization for the rest of `optType` — but you can explicitly specify whether you want minimization or maximization. You can also optimize properties to the target value (e.g., band gaps close to 1.34 eV are interesting for photovoltaics).

Possible values (characters):

Name	Number	Description
enthalpy	1	to find the stable phases
volume	2	minimization of volume per atom (to find the densest structure)
hardness	3	hardness maximization (to find the hardest phase)
struc_order	4	maximization of the degree of order (to find the most ordered structure)
density	5	maximization of density
diel_sus	6	maximization of the static dielectric susceptibility (only for VASP and GULP)
bandgap	7	maximization of the band gap (only for VASP)
diel_gap	8	maximization of electrical energy storage capacity (only for VASP)
mag_moment	9	maximization of the magnetization (i.e. magnetic moment per volume) (only for VASP)
quasientropy	10	maximization of structural quasientropy
birefringence	11	difference between largest and smallest eigenvalues of the refractive index tensor (only for VASP)
ZT	14	thermoelectric figure of merit ZT (only for VASP and BoltzTraP)
Fphon	17	free energy at finite temperature

Elasticity-related properties ("11**"):

Value	Number	Description
K, Bulk Modulus	1101	maximization of bulk modulus
G, Shear Modulus	1102	maximization of shear modulus
E, Young's Modulus	1103	maximization of Young's modulus
v, Poisson's ratio	1104	maximization of Poisson's ratio
G/K, Pugh's modulus ratio	1105	maximization of Pugh's modulus ratio
Hv, Vickers hardness	1106	maximization of Vickers hardness
Kg, Fracture toughness	1107	maximization of fracture toughness
D, Debye temperature	1108	maximization of Debye temperature
Vm, sound velocity	1109	maximization of sound velocity
S-wave velocity	1110	maximization of S-wave velocity
P-wave velocity	1111	maximization of P-wave velocity

Note: Elasticity-related properties are supported only for VASP (starting from VASP 5.1) and GULP. For VASP users, you need to add one more `INCAR_*` file to the `Specific/` folder with the parameters `IBRION`=6, `ISIF`$\geq$3 and `NFREE`=4. The estimates of bulk, shear and Young's moduli are the Voigh-Reuss-Hill (VRH) averages. The Vickers hardness is calculated with the Chen-Niu model[19]. Fracture toughness is calculated using the Niu-Niu-Oganov model[?] .

Default: `enthalpy`

Format:

```
% optType
enthalpy (equivalent to Min_enthalpy)
% EndOptType
```

another example:

```
% optType
Min_(bandgap-1.34)^2
% EndOptType
```

Note: In the latter case, we optimize a mathematical expression as the `optType` variable. The mathematical expression should be in parentheses and should be a valid MATLAB expression. The whole expression (including min_ and max_ if applicable) should be one unit, do NOT put blank space between the expression compartments. For example "(bandgap - 1.3) ^2"or min_ (bandgap-3) are NOT valid formats.

Multiobjective (Pareto) optimizations are also available by specifying several fitnesses in the optType keyblock, e.g.,

```
% optType
3 1
% EndOptType
```

or

```
% optType
max_Hardness enthalpy
% EndOptType
```

In multiobjective optimization, it is useful to start with a few generations of enthalpy-only optimization and then switch to Pareto optimization of all desired properties. The number of initial generations spent on enthalpy-only optimization by default is zero and is given in brackets as follows:

```
% optType
3 6 [5]
% EndOptType
```

or

```
% optType
max_Hardness min_Diel_sus [5]
% EndOptType
```

This means USPEX does enthalpy-only optimization up to the 5th generation, and then switches to multiobjective optimization.

Fig. 6a gives an example of hardness maximization for TiO_2 (`optType`=hardness), showing maximum possible hardness 14 GPa[20] and refuting claims of Dubrovinsky (2001) about ultrahardness of TiO_2[21], a good example of how a simple USPEX run can resolve a long-standing dispute. Fig. 6b shows an example of Pareto optimization, clearly displaying frequent trade-off between stability and properties.

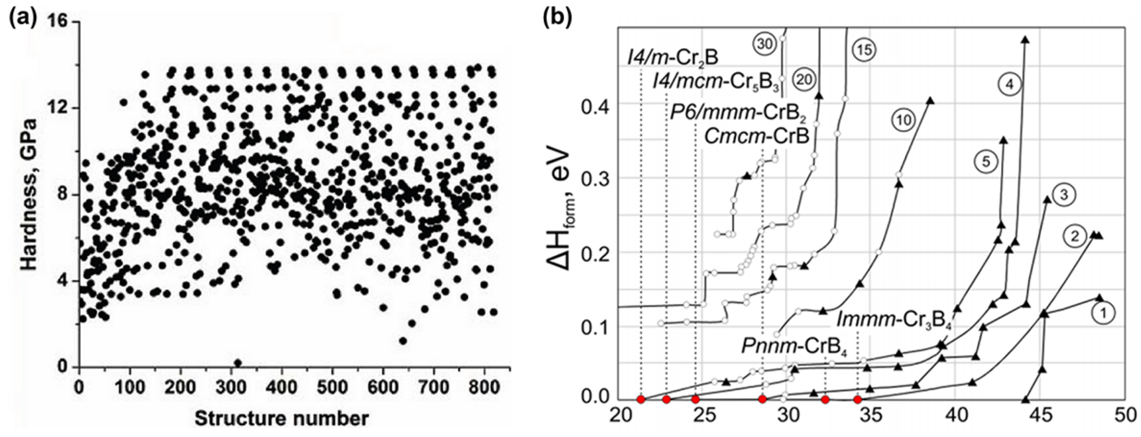


Figure 5: **Examples of properties optimizations: (a) Prediction of the hardest structure of TiO_2[20]. (b) Pareto optimization of hardness and stability in the Cr-B system, showing several Pareto fronts[22].**

Notes: If `optType`=`bandgap` or `diel_gap`, instead of the gap we use an extended function that also behaves continuously for metals — namely, $\Delta E_g - g(E_F)/N$, where ΔE_g is the gap, $g(E_F)$ is the density of states at the Fermi level (for metals) and N is the number of atoms in the unit cell. Thanks to the continuity of this function, global maximization of gap-related quantities can even be performed for metallic solutions. For metals it is equal to the DOS at the Fermi level, for semiconductors and insulators — to the band gap.

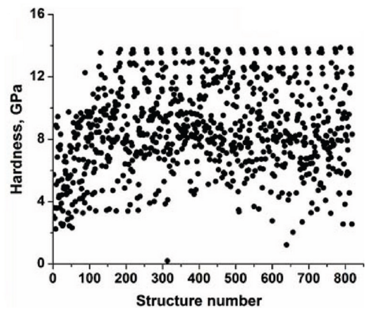


Figure 6: **Predictions of the hardest structure of TiO_2.**

▷ *variable* `atomType`

Meaning: Describes the identity of each type of atom.

Default: none, must specify explicitly

Format:

If you prefer to use the atomic numbers from Mendeleev's Periodic Table of the Elements, specify:

```
% atomType
12 14 8
% EndAtomType
```

Or, if you prefer to use atomic names, specify:

```
% atomType
Mg Si O
% EndAtomType
```

You can alternatively specify the full names of the elements, for example:

```
% atomType
Magnesium Silicon Oxygen
% EndAtomType
```

▷ *variable* `numSpecies`

Meaning: Specifies the number of atoms of each type.

Default: none, must specify explicitly

Format:

```
% numSpecies
4 4 12
% EndNumSpecies
```

This means there are 4 atoms of the first type, 4 of the second type, and 12 of the third type.

Notes: For variable-composition calculations, you have to specify the compositional building blocks as follows:

```
% numSpecies
2 0 3
0 1 1
% EndNumSpecies
```

This means that the first building block has formula A_2C_3 and the second building block has formula BC, where A, B and C are described in the block `atomType`. All structures will then have the formula $xA_2C_3 + yBC$ with x, y = (0,1,2,...) — or $A_{2x}B_yC_{3x+y}$. If you want to do prediction of all possible compositions in the A-B-C system, you should specify:

```
% numSpecies
1 0 0
0 1 0
0 0 1
```

```
% EndNumSpecies
```

You can also do fixed-composition calculations with a variable number of formula units; in this case set `calculationType`=300, the composition of one formula unit, for example, A_2BC_4:

```
% numSpecies
2 1 4
% EndNumSpecies
```

and minimum and maximum total numbers of atoms in the unit cell, for example:

```
14 :  minAt
28 :  maxAt
```

▷ *variable* `magRatio`

Meaning: Initial ratio of structures with different magnetic orders, namely NM, FM-LS, FM-HS, AFM-L, AFM-H, FM-LH, AF-LH states, respectively. Only VASP is supported.

Default: `0.1, 0.9/4, 0.9/4, 0.9/4, 0.9/4, 0, 0`

Format:

```
% magRatio
1/8 1/8 1/8 1/8 1/8 0 0
% EndMagRatio
```

This means that probabilities of generating NM, FM-LS, FM-HS, AFM-L, AFM-H structures are all 20% (**NOT 1/8 here! The number are rescaled so that sum of probabilities is 1.**). No structures with FM-LH, AF-LH magnetic states will be generated.
Notes:
(1) The sum of `magRatio` can be larger than 1, the ratio will be rescaled to 1 automatically.

(2) The meaning and initial magnetic moment value in USPEX:

- NM — non-magnetic;
- FM-LS — low-spin ferromagnetic;
- FM-HS — high-spin ferromagnetic;
- AFM-L — low-spin antiferromagnetic;
- AFM-H — high-spin antiferromagnetic;
- FM-LH — low/high spin mixed ferromagnetic;
- AF-LH — low/high spin mixed antiferromagnetic.

(3) For NM states, the initial magnetic moment will be 0 for all atoms. The initial magnetic moment `MAGMOM` of the atoms will be set to 1 and 4 for low and high spin states, respectively. For low/high spin mixed states, `MAGMOM` will be set to 1 or 4 randomly for each atom.

(4) AFM type structures will not be generated when having odd number of magnetic atoms in a unit cell.

(5) `magRatio` is also used for the mutation ratio in spinmuation operation.

▷ *variable* `ldaU`

Meaning: Specifies Hubbard **U** value for atoms of each type with the LDA+U method. Only VASP is supported.

Default: `0` for each type of atoms

Format:

```
% ldaU
4 0
% EndLdaU
```

▷ *variable* `ExternalPressure`

Meaning: Specifies external pressure at which you want to find structures, in GPa.

Default: `0`

Format:

```
100 :  ExternalPressure
```

Note: As of USPEX version 9.4.1 pressure value (in GPa) is set by the tag `ExternalPressure` in the `INPUT.txt` file. **Please:** do not specify it in relaxation files in the `Specific/` folder.

▷ *variable* `valences`

Meaning: Describes the valences of each type of atom. Used only to evaluate bond hardnesses, which are used for computing the approximate dynamical matrix (for softmutation) and hardness of the crystal.

Default: USPEX has a table of default valences (see Appendix 9.9). Beware, however, that for some elements (*e.g.*, N, S, W, Fe, Cr, *etc.*) many valence states are possible. Unless you calculate hardness, just use the default values by not specifying valences. If you do calculate the hardness, you need to carefully and explicitly specify the valence explicitly.

Format:

```
% valences
2 4 2
% EndValences
```

▷ *variable* `goodBonds`

Meaning: Specifes, in the matrix form, the minimum bond valences for contacts that will be considered as important bonds. Like the `IonDistances` matrix (see below), this is a square matrix. This is only used in calculations of hardness and in softmutation. One can estimate these values for a given bond type taking `goodBonds`$=\frac{valence}{max\_coordination\_number}$ or slightly smaller.

Default: USPEX can make a reasonable default estimation of `goodBonds`, you will see the values in `OUTPUT.txt` file. This should be sufficient for most purposes, but for hardness calculations you may need to carefully examine these values and perhaps set them manually. For more details, see Appendix 9.10

Format:

```
% goodBonds
10.0 10.0 0.2
10.0 10.0 0.5
0.2 0.5 10.0
% EndGoodBonds
```

Notes: The dimensionality of this matrix must be equal to either the number of atomic species or unity. If only one number is used, the matrix is filled with this number. The above matrix reads as follows: to be considered a bond, the Mg–Mg distance should be short enough to have bond valence of 10 or more, the same for Mg–Si, Si–Si, and O–O bonds (by using such exclusive criteria, we effectively disregard these interactions from the softmutation and hardness calculations), whereas Mg–O bonds that will be considered for hardness and softmutation calculations will have a bond valence of 0.2 or more, and the Si–O bonds will have a bond valence of 0.5 or more.

▷ *variable* `checkMolecules`

Meaning: Switches on/off post-relaxation check that original molecules (files `MOL_1`, `MOL_2`, ...) are intact. Useful for molecular crystals (`calculationType`=310, 311).

Possible values (integer):

- 0 — check is not performed, structures with broken or merged molecules are considered. (We strongly suggest users not to use this.)
- 1 — check is performed, all the structures with broken or merged molecules are discarded.

Default: `1`

Format:

```
1 :  checkMolecules
```

▷ *variable* `checkConnectivity`

Meaning: Switches on/off hardness calculation and connectivity-related criteria in softmutation.

Possible values (integer):

- 0 — connectivity is not checked, no hardness calculations;
- 1 — connectivity is taken into account, hardness is calculated.

Default: `0`

Format:

```
1 :  checkConnectivity
```

▷ *variable* `fitLimit`

Meaning: USPEX stops when it finds a fitness value equal to or better than `fitLimit`

Default: no default, has to be specified by the user.

Format:

```
-9.319 :  fitLimit
```

4.2 Population

▷ *variable* `populationSize`

Meaning: The number of structures in each generation; size of initial generation can be set separately, if needed.

Default: $2 \times N$ rounded to the closest 10, where N is the number of atoms/cell (or `maxAt` for variable composition). The upper limit is 60. Usually, you can trust these default settings.

Format:

```
20 :  populationSize
```

▷ *variable* `initialPopSize`

Meaning: The number of structures in the initial generation.

Default: equal to `populationSize`.

Format:

```
20 :  initialPopSize
```

Note: In most situations, we suggest that these two parameters be equal. Sometimes (especially in variable-composition calculations) it may be useful to specify `initialPopSize` to be larger than `populationSize`. It is also possible to have a smaller `initialPopSize`, if one wants to produce the first generation from seed structures.

▷ *variable* `numGenerations`

Meaning: Maximum number of generations allowed for the simulation. The simulation can terminate earlier, if the same best structure remains unchanged for `stopCrit` generations.

Default: `100`

Format:

```
    50 :  numGenerations
```

▷ *variable* `stopCrit`

Meaning: The simulation is stopped if the best structure did not change for stopCrit generations, or when `numGenerations` have expired — whichever happens first.

Default: total number of atoms for fixed-composition runs, maximum number of atoms `maxAt` for variable-composition runs.

Format:

```
    20 :  stopCrit
```

4.3 Survival of the fittest and selection

▷ *variable* `bestFrac`

Meaning: Fraction of the current generation that shall be used to produce the next generation.

Default: `0.7`

Format:

```
    0.7 :  bestFrac
```

Note: This is an important parameter, values between 0.5–0.8 are reasonable.

▷ *variable* `keepBestHM`

Meaning: Defines how many best structures will survive into the next generation.

Default: `0.15`×`populationSize`

Format:

```
    3 :  keepBestHM
```

▷ *variable* `reoptOld`

Meaning: Defines re-relaxation of the survived structures. If `reoptOld`=0, these structures will be unchanged while if `reoptOld`=1, they will be re-relaxed again. Usually `reoptOld`=0 is a reasonable choice (provided your structure relaxation was high quality).

Default: `0`

Format:

```
1 :  reoptOld
```

4.4 Structure generation and variation operators

▷ *variable* `symmetries`

Meaning: Possible symmetry groups for random symmetric structure generator crystals (spacegroups), layer plane groups for 2D-crystals, plane groups for surfaces, or point groups for clusters. A certain number of structures will be produced using randomly selected groups from this list, using randomly generated lattice parameters and atomic coordinates. During this process special Wyckoff sites can be produced from general positions (Fig. 7).

Default:

- For 3D crystals: `2-230`
- For 2D crystals: `2-80`
- For clusters: `E C2 D2 C4 C3 C6 T S2 Ch1 Cv2 S4 S6 Ch3 Th Ch2 Dh2 Ch4 D3 Ch6 O D4 Cv3 D6 Td Cv4 Dd3 Cv6 Oh Dd2 Dh3 Dh4 Dh6 Oh C5 S5 S10 Cv5 Ch5 D5 Dd5 Dh5 I Ih`

Format:

```
% symmetries
195-198 200 215-230
% EndSymmetries
```

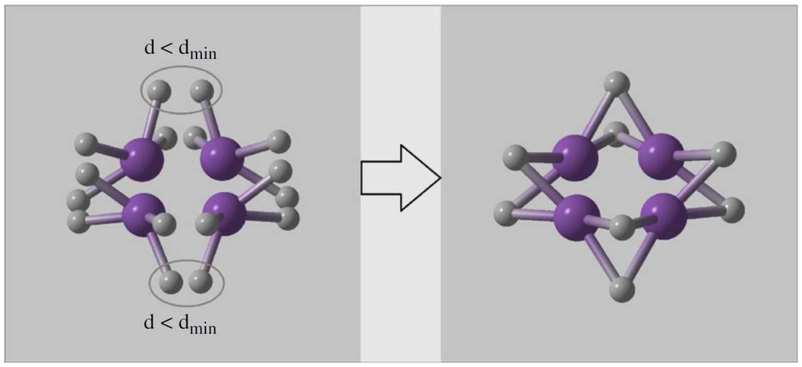


Figure 7: **Example of random symmetric structure generation and merging atoms onto special Wyckoff positions (for detail, see Ref.[13]).**

▷ *variable* `splitInto`

Meaning: Defines the number of identical subcells or pseudosubcells in the unit cell. If you do not want to use splitting, just use the value 1, or delete the block. Use splitting only for systems with >25–30 atoms/cell.

Default: `1`

Format:

```
% splitInto (number of subcells into which the unit cell is split)
1 2 4
% EndSplitInto
```

Subcells introduce extra translational (pseudo)symmetry. In addition to this, each subcell can be built using a special space groups algorithm developed by A.R. Oganov and H.T. Stokes and implemented by H.T. Stokes (see Reference[13]).

▷ *variable* `fracGene`

Meaning: Percentage of structures obtained by heredity; 0.5 means 50%, *etc.*

Default: `0.5`

Format:

```
0.5 :  fracGene
```

▷ *variable* `fracRand`

Meaning: Fraction of the generation produced by random symmetric structure generator.

Default: `0.2`

Format:

```
0.20 :  fracRand
```

▷ *variable* `fracTopRand`

Meaning: Percentage of structures obtained by topological random generator.

Default: `0.2`

Format:

```
0.20 :  fracTopRand
```

▷ *variable* `fracPerm`

Meaning: Percentage of structures obtained by permutation; 0.1 means 10%, *etc.*

Default: `0.1` if there is more than one type of atom/molecule; `0` otherwise.

Format:

```
0.1 :  fracPerm
```

▷ *variable* `fracAtomsMut`

Meaning: Specifies the percentage of structures obtained by softmutation or coormutation.

Default: `0.1`

Format:

```
0.1 :  fracAtomsMut
```

▷ *variable* `fracRotMut`

Meaning: Percentage of structures obtained by mutating orientations of randomly selected molecules; 0.1 means 10%, *etc.* Use it only for molecular crystal structure prediction (`calculationType` = 310, 311)

Default: `0.1` for molecular crystals; `0` otherwise.

Format:

```
0.1 :  fracRotMut
```

▷ *variable* `fracLatMut`

Meaning: Percentage of structures obtained from lattice mutations; 0.1 means 10%, *etc.*

Default: `0` for fixed-cell prediction; `0.1` otherwise. We tend to se always `fracLatMut` = 0

Format:

```
0.1 :  fracLatMut
```

Note: If the sum of all the fractions (`fracGene` + `fracRand` + `fracPerm` + ...) is not

equal to 1, all fractions will be normalized.

▷ *variable* `fracSpinMut`

Meaning: Percentage of structures obtained by spin mutation; 0.1 means 10%, *etc.* Used only for magnetic structure optimization.

Default: `0.1`

Format:

```
0.2 : fracSpin
```

Notes:

(1) Spin mutaion operator always changes the magnetic structure. For example, FM-L states will **never** generate FM-L states after mutation, but can transform to NM, FM-H and so on. For AFM states, spin mutation can generate another AFM, but with different spin-up and spin-down arrangement.

(2) The ratio of mutation to new states is determined by `magRatio`.

▷ *variable* `howManySwaps`

Meaning: For permutation, the number of pairwise swaps will be randomly drawn from a uniform distribution between 1 and `howManySwaps`.

Default: `0.5`×(maximum number of possible swaps). If atoms Na and Nb, and atoms Nc and Nd are swappable, then the total number of possible swaps is $\min(Na, Nb) + \min(Nc, Nd)$, and the default for `howManySwaps` is $0.5 \times [\min(Na, Nb) + \min(Nc, Nd)]$. In most cases, it is a good idea to rely on this default.

Format:

```
5 : howManySwaps
```

▷ *variable* `specificSwaps`

Meaning: Specifies which atom types you allow to swap in permutation.

Default: blank line, which means no specific swaps and all atoms are permutable.

Format:

```
% specificSwaps
1 2
% EndSpecific
```

Note: In this case, atoms of type 1 can be swapped with atoms of type 2. If you want to try all possible swaps, just leave a blank line inside this keyblock, or delete the block.

▷ *variable* `AutoFrac`

Meaning: Enables automatic evolution of percentages of variation operators, which speeds up the calculation by up to ∼2 times. This is done by encouraging operators which produce more diverse and lower-energy structures. To switch to user-defined percentages, set `AutoFrac`=0.

Default: `0`

Format:

```
    1 :  AutoFrac
```

4.5 Constraints

The same structure can be represented in an infinite number of coordinate systems ("modular invariance"). Many of these equivalent choices will lead to very flat unit cells, which creates problems for structure relaxation and energy calculation (*e.g.*, a very large number of k-points are needed). The constraint, well known in crystallography, that the cell angles be between 60° and 120°, does not remove all redundancies and problematic cells (*e.g.*, thus allowed cells with $\alpha = \beta = \gamma \sim 120°$ are practically flat). Therefore, we developed[20;23] a scheme to obtain special cell shapes with the shortest cell vectors. This transformation can be performed if there is at least one lattice vector whose projection onto any other cell vector or the diagonal vector of the opposite cell face is greater (by modulus) than half the length of that vector, *i.e.*, for pairs **a** and **b**, or **c** and $(\mathbf{a}+\mathbf{b})$ these criteria are:

$$\left|\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{b}|}\right| > \frac{|\mathbf{b}|}{2} \tag{2}$$

$$\left|\frac{\mathbf{a}\cdot\mathbf{b}}{|\mathbf{a}|}\right| > \frac{|\mathbf{a}|}{2} \tag{3}$$

$$\left|\frac{\mathbf{c}\cdot(\mathbf{a}+\mathbf{b})}{|\mathbf{c}|}\right| > \frac{|\mathbf{c}|}{2} \tag{4}$$

$$\left|\frac{\mathbf{c}\cdot(\mathbf{a}+\mathbf{b})}{|\mathbf{a}+\mathbf{b}|}\right| > \frac{|\mathbf{a}+\mathbf{b}|}{2} \tag{5}$$

For instance, for the criterion 2 the new vector $\mathbf{a}^*$ equals:

$$\mathbf{a}^* = \mathbf{a} - \text{ceil}\left(\frac{|\mathbf{a}\cdot\mathbf{b}|}{|\mathbf{b}|^2}\right)\text{sign}(\mathbf{a}\cdot\mathbf{b})\mathbf{b} \tag{6}$$

This transformation is performed iteratively, completely avoids pathological cell shapes, and thus solves the problem. During this transformation, the atomic fractional coordinates are transformed so that the original and transformed structures are identical (during the transformation, the Cartesian coordinates of the atoms remain invariant).

▷ *variable* `minVectorLength`

Meaning: Sets the minimum length of a cell parameter of a newly generated structure.

Default: 1.8 × covalent diameter of the largest atom. For molecular crystals (`calculationType` = 310, 311) default value is 1.8 × max(`MolCenters`).

Format:

```
2.0 :  minVectorLength
```

Commonly used computational methods (pseudopotentials, PAW, LAPW, and many parametric forcefields) fail when the interatomic distances are too small. This situation needs to be avoided by specifying the minimum distances between each pair of atoms using the `IonDistances` square matrix:

▷ *variable* `IonDistances`

Meaning: Sets the minimum interatomic distance matrix between different atom types. Structures with distances lower than `IonDistances` will be strictly discarded.

Default: the `IonDistances` between atom A and B are estimated as $0.22 \times (V_A^{1/3} + V_B^{1/3})$ but not larger than 1.2 Å, and $0.45 \times (V_A^{1/3} + V_B^{1/3})$ in molecular calculations, where V_A and V_B are the default volumes of atom A and B estimated in USPEX.

Format:

```
% IonDistances
1.0 1.0 0.8
1.0 1.0 0.8
0.8 0.8 1.0
% EndDistances
```

Note: The dimensions of this matrix must be equal to the number of atomic species. If the compound in the example above is $MgSiO_3$, the matrix reads as follows: the minimum Mg–Mg distance allowed in a newly generated structure is 1.0 Å, the minimum Mg–Si, Si–Si and O–O distances are also 1.0 Å, and the minimum Mg–O and Si–O distances are 0.8 Å. You can use this keymatrix to incorporate further system-specific information: *e.g.*, if you know that Mg atoms prefer to be very far apart and are never closer than 3 Å in your system, you can specify this information. Beware, however, that the larger these minimum distances, the more difficult it is to generate structures fulfilling these constraints (especially for large systems), so strive for a compromise and remember that `IonDistances` must be **much** smaller than the actual distances in the crystal: realistic distances will be achieved by structure relaxation. What `IonDistances` trick does is to avoid structures which cannot be relaxed correctly.

▷ *variable* `constraint_enhancement`

Meaning: Allows one to apply the stricter constraints of the `IonDistances` matrix (by `constraint_enhancement` times) for symmetric random structures (for all variation operators, unenhanced `IonDistances` matrix still applies). Only use it if you know what you are doing.

Default: `1`

Format:

```
1 : constraint_enhancement
```

For molecular crystals, the following keyblock is extremely important:

▷ *variable* `MolCenters`

Meaning: Matrix of minimal distances between the geometric centers of molecules. Any distances lower than these indicate large overlap of the molecules, are unphysical and will be strictly avoided.

Default: none.

Format:

```
% MolCenters
5.5 7.7
0.0 9.7
% EndMol
```

Note: In the above example, there are two types of molecules. In all of the generated structures, the distance between the geometric centers of the molecules of the first type must be at least 5.5 Å (A–A distance), the distance between the centers of the molecules of the first and second type — 7.7 Å (A–B distance), and the distance between the molecules of the second type — 9.7 Å (B–B distance).

4.6 Cell

It is useful to create all new structures (before relaxing them) with a unit cell volume appropriate for given conditions. This can be specified in the `Latticevalues` keyblock:

▷ *variable* `Latticevalues`

Meaning: Specifies the initial volume of the unit cell or known lattice parameters.

Default: For cell volumes you don't have to specify values — USPEX has a powerful algorithm to make reasonable estimates at any pressure.

Format:

```
% Latticevalues
125.00
```

```
% Endvalues
```

Notes: (1) This volume is only used as an initial guess to speed up structure relaxation and does not affect the results, because each structure is fully optimized and adopts the volume corresponding to the (free) energy minimum. This keyblock also has another use: when you know the lattice parameters (*e.g.*, from experiment), you can specify them in 3×3 matrix (calculationType = 300/310) or 2×2 matrix (-200) in the `Latticevalues` keyblock instead of unit cell volume, *e.g.*:

```
% Latticevalues
7.49 0.0 0.0
0.0 9.71 0.0
0.0 0.0 7.07
% Endvalues
```

Alternatively, you can specify unit cell parameters just by listing `a`, `b`, `c`, α, β, and γ values:

```
% Latticevalues
10.1 8.4 12.5 90.0 101.3 90.0
% Endvalues
```

For 2D crystal (calculationType = -200), you just need cell parameters a, b, and α.

```
% Latticevalues
10.1 8.4 90.0
% Endvalues
```

Attention: if you do a calculation with a fixed monoclinic cell, please use setting with special angle β (standard setting).

(2) For variable-composition calculations, you have to specify the volume of end members of the compositional search space, *e.g.*:

```
% Latticevalues
12.5 14.0 11.0
% Endvalues
```

(3) Users no longer need to specify the unit cell or atomic volumes in the keyblock `Latticevalues` — a special algorithm has been implemented that accurately estimates it at the pressure of interest, without the need for the user to specify it. This option works well and is available for any `calculationType` where input volumes are required: 3**, 2D-crystals, 110, 000. You can also use online program `http://uspex-team.org/online_utilities/volume_estimation`. The users can also input the volumes manually.

(4) If you study molecular crystals under pressure, you might sometimes need to increase the initial volumes somewhat, in order to be able to generate initial random structures.

4.7 Restart

If something goes wrong, you may want to continue the calculation from the point where it stopped — or from an earlier point. If all you want to do is continue the run from where it stopped, you do not need to change any settings.

If you want to restart from a particular generation in a particular `results`-folder, then specify `pickUpGen` = number of the generation from which you want to start, `pickUpFolder` = number of `results`-folder (*e.g.*, 1 for `results1`, 2 for `results2`, ...) from which the restart needs to be. If `pickUpGen`=0, then a new calculation is started. The default values for both parameters are 0. For example, to restart a calculation performed in the folder `results5` from generation number 10, specify:

```
10 :  pickUpGen
5 :  pickUpFolder
```

4.8 Details of *ab initio* calculations

USPEX employs a powerful two-level parallelization scheme, making its parallel scalability exemplary. The first level of parallelization is performed within structure relaxation codes, the second level of parallelization distributes the calculation over the individuals in the same population (since structures within the same generation are independent of each other).

First, you must specify which code(s) you want to use for structure relaxation and fitness calculation:

▷ *variable* `abinitioCode`

Meaning: Defines the code used for every optimization step.

Default: `1` for every optimization step (VASP)

Format:

```
% abinitioCode
3 2 2 1 1
% ENDabinit
```

Note: Numbers indicate the code used at each step of structure relaxation:

1 — VASP	5 — ORCA
2 — SIESTA	6 — DMACRYS
3 — GULP	7 — CP2K
4 — LAMMPS	8 — Quantum Espresso

`9` — FHI-aims	`13` — MOPAC
`10` — ATK	`14` — BoltzTraP
`11` — CASTEP	`15` — DFTB
`12` — Tinker	`16` — Gaussian (only for clusters)

▷ *variable* `KresolStart`

Meaning: Specifies the reciprocal-space resolution for k-points generation (units: $2\pi\text{Å}^{-1}$).

Default: from 0.2 to 0.08 linearly

Format:

```
 % KresolStart
0.2 0.16 0.12 0.08
% Kresolend
```

Note: You can enter several values (one for each step of structure relaxation), starting with cruder (*i.e.*, larger) values and ending with high resolution. This dramatically speeds up calculations, especially for metals, where very many k-points are needed. This keyblock is important for ab-initio calculations (through some codes, e.g. VASP and SIESTA, now have similar tricks)).

For **clusters**, **2D-crystals**, and **surfaces**, you have to specify the thickness of the vacuum region around the cluster (or around the surface slab):

▷ *variable* `vacuumSize`

Meaning: Defines the amount of vacuum added around the structure (closest distance in Å between neighboring clusters in adjacent unit cells). Used only for surfaces, 2D-crystals, and nanoparticles.

Default: `10` Å for every step of relaxation

Format:

```
% vacuumSize
10 10 15 20 20
% EndVacuumSize
```

▷ *variable* `numParallelCalcs`

Meaning: Specifies how many structure relaxations you want to run in parallel.

Default: `1`

Format:

```
10 :  numParallelCalcs
```

You need to supply the job submission files or the names of executable files for each code/mode you are using.

▷ *variable* `commandExecutable`

Meaning: Specifies the name of the job submission files or executables for a given code.

Default: no default, has to be specified by the user.

Format:

```
% commandExecutable
gulp < input > output
mpirun -np 8 vasp > out
mpirun -np 8 vasp > out
mpirun -np 8 vasp > out
% EndExecutable
```

Note: Every line corresponds to a stage of relaxation — the first line describes the execution of the first stage of relaxation, *etc.* For example, `abinitioCode` equal to "`3 1 1 1`" means that the first relaxation step will be performed with GULP, while the subsequent steps will be performed using VASP via the command "`mpirun -np 8 vasp > out`". If only one line is present in `commandExecutable`, then the same execution will be performed for all steps of relaxation.

You can actually use USPEX on practically any platform in the remote submission mode. In that case, your workstation will prepare input (including jobs), send them to the remote compute nodes, check when the calculations are complete, get the results back, analyze them, and prepare new input. The amount of data being sent to and fro is not large, so the network does not need to be very fast. Job submission is, of course, machine-dependent.

▷ *variable* `whichCluster`

Meaning: Specifies the types of job submission.

Possible values (integer):

- 0 — no-job-script;
- 1 — local submission;
- 2 — remote submission.

Default: `0`

Format:

```
1 :  whichCluster
```

▷ *variable* remoteFolder

Meaning: Folder on the remote supercomputer where the calculation will be performed. This keyword is activated only if `whichCluster`=2.

Default: none

Format:

```
Blind_test : remoteFolder
```

Note: there is a similar parameter specified in the remote submission file — `homeFolder`. The actual path to the calculation will be `/*remoteFolder*/CalcFolderX` where `X`=1, 2, 3,. . . .

▷ *variable* PhaseDiagram

Meaning: Enables calculation of a phase diagram for `calculationType`=300 and 301. It gives an idea (crude one — just to get a rough idea!) of which structures may be stable at higher and lower pressures, and a rough idea of transition pressures.

Default: 0

Format:

```
1 : PhaseDiagram
```

4.9 Fingerprint settings

Please read (Oganov & Valle, 2009[17]) for details on fingerprint functions.

▷ *variable* RmaxFing

Meaning: Distance cutoff (in Å).

Default: 10.0

Format:

```
10.0 : RmaxFing
```

▷ *variable* deltaFing

Meaning: Discretization (in Å) of the fingerprint function.

Default: 0.08

Format:

```
0.10 : deltaFing
```

▷ *variable* sigmaFing

Meaning: Gaussian broadening of interatomic distances.

Default: `0.03`

Format:

```
    0.05 : sigmaFing
```

`toleranceFing` (default=0.008) specifies the minimal cosine distances between structures that qualify them as non-identical — for participating in the production of child structures and for survival of the fittest, respectively. This depends on the precision of structure relaxation and the physics of the system (for instance: for alloy ordering problems, fingerprints belonging to different structures will be very similar, and these tolerance parameters should be made small).

4.10 Antiseed settings

A family of antiseed techniques have been developed and implemented in USPEX, all based on the idea of penalizing already sampled structures to ensure that the simulation is not trapped in a local minimum. Here, time-dependent fitness is the sum of the actual enthalpy (or another fitness property of interest) and a history-dependent term, which is the sum of the Gaussian potentials added to already sampled parts of the energy landscape:

$$f = f_0 + \sum_a W_a \exp\left(-\frac{d_{ia}^2}{2\sigma_a^2}\right),$$

where f is fitness (f_0 — the true fitness, f — history-dependent fitness), W_a is the height and σ_a is the width of the Gaussian. In our approach, Gaussian parameters change depending on the population diversity and energy spread at each generation.

There are three ways to use this technique. In the first, you can put the structure that you wish to penalize in the `AntiSeeds` folder. For example, this can be the ground state structure — in this case, USPEX will try to find the second lowest-enthalpy structure.

In the second and third methods, you don't specify antiseed structure(s) — the calculation either uses all sampled structures as antiseeds (well tested; the recommended approach) or just the best structure in each generation. You need to specify a few settings:

▷ *variable* `antiSeedsActivation`

Meaning: Specifies from which generation the antiseed mode will be switched on. When antiSeedsActivation $= N > 0$, Gaussians are added to all structures starting from generation N, and when $N < 0$ — Gaussians are only added to the best structure of each generation, starting from generation N. When $N = 0$, Gaussians are only added to the structures put in the `AntiSeeds` folder. If you don't want to use antiseeds, specify very large `antiSeedsActivation` (for example, 5000) and `antiSeedsMax`=0.0.

Default: `5000`

Format:

```
1 :  antiSeedsActivation
```

▷ *variable* antiSeedsMax

Meaning: Specifies the height of the Gaussian, in units of the mean square deviation of the enthalpy in the generation (computed only among `bestFrac` structures, *i.e.*, among potential parents). We recommend `antiSeedsMax`=0.01.

Default: `0.000`

Format:

```
0.005 :  antiSeedsMax
```

▷ *variable* antiSeedsSigma

Meaning: Specifies the width of the Gaussian, in units of the average distance between structures in the generation (computed only among `bestFrac` structures, *i.e.*, among potential parents). We recommend `antiSeedsSigma`=0.005.

Default: `0.001`

Format:

```
0.005 :  antiSeedsSigma
```

Fig. 8 shows an example of use of antiseed technique.

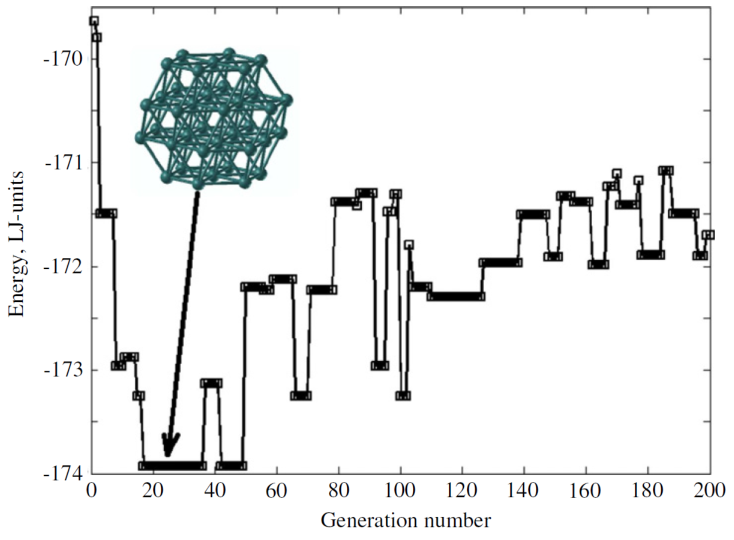


Figure 8: **Example of a calculation of a Lennard-Jones cluster with 38 atoms with the use of antiseeds.** The energy of the best structure in every generation is plotted. One can clearly see that the algorithm does not get stuck for a long time to any of the candidate minima and quickly finds the ground state. Here we used `antiSeedsActivation`=1, `antiSeedsMax`=0.01, `antiSeedsSigma`=0.001.

4.11 Space group determination

▷ *variable* doSpaceGroup

Meaning: Determines space groups and also writes output in the crystallographic `*.CIF`-format (this makes your life easier when preparing publications, but beware that space groups may often be under-determined if relaxation was not very precise and if very stringent tolerances were set for the symmetry finder). This option is enabled thanks to the spglib-library https://atztogo.github.io/spglib/ code.

Default: `1`, if `calculationType=3**` (`300`, `301`, `310`, `311` — bulk crystals) and `0` otherwise.

Format:

```
1 : doSpaceGroup (0 - no space groups, 1 - determine space groups)
```

▷ *variable* SymTolerance

Meaning: Precision for symmetry determination using the symmetry finder code. Can be specified either as a number (in Å) or as `high | medium | low` (= `0.05 | 0.10 | 0.20`)

Default: `medium`

Format:

```
medium : SymTolerance
```

4.12 Keywords for developers

▷ *variable* repeatForStatistics

Meaning: Number of automatically executed USPEX runs. USPEX simulations are stochastic, and redoing the simulation with the same input parameters does not necessarily yield the same results. While the final result — the ground state — is the same (hopefully!), the number of steps it takes to reach it and the trajectory in configurational space will differ from run to run. To quantify performance of the algorith, you MUST collect some statistics — do not rely on just a single run (which may be lucky or unlucky... USPEX does not rely on luck!). This option is only of interest to developers and it only makes sense to collect statistics with forcefields (*e.g.*, using GULP).

Default: `1` (*i.e.*, no statistics will be gathered)

Format:

```
20 : repeatForStatistics
```

▷ *variable* `stopFitness`

Meaning: Specifies the fitness value so that the calculation will stop after reaching fitness $\leq$ `stopFitness`.

Default: no default, has to be specified by the user.

Format:

```
90.912 : stopFitness
```

Note: Automatic analysis of statistics is enabled when `stopFitness` is specified. It is recommended to set `repeatForStatistics` keyword to values >1 to collect statistics of reachability of `stopFitness`. Sample output is:

```
Number of files to be processed: 20
Target enthalpy: 90.912

Generation: 23    Number: 1326    Enthalpy:    90.9119    Mat-file: /home/USPEX/01/results1/USPEX.mat
Generation: 22    Number: 1224    Enthalpy:    90.9119    Mat-file: /home/USPEX/02/results1/USPEX.mat
Generation: 60    Number: 3451    Enthalpy:    90.9119    Mat-file: /home/USPEX/03/results1/USPEX.mat
Generation: 30    Number: 1739    Enthalpy:    90.9119    Mat-file: /home/USPEX/04/results1/USPEX.mat
Generation: 17    Number:  956    Enthalpy:    90.9119    Mat-file: /home/USPEX/05/results1/USPEX.mat
Generation: 36    Number: 2055    Enthalpy:    90.9119    Mat-file: /home/USPEX/06/results1/USPEX.mat
Generation: 35    Number: 1987    Enthalpy:    90.9119    Mat-file: /home/USPEX/07/results1/USPEX.mat
Generation: 22    Number: 1241    Enthalpy:    90.9119    Mat-file: /home/USPEX/08/results1/USPEX.mat
Generation: 18    Number: 1002    Enthalpy:    90.9119    Mat-file: /home/USPEX/09/results1/USPEX.mat
Generation: 29    Number: 1641    Enthalpy:    90.9119    Mat-file: /home/USPEX/10/results1/USPEX.mat
Generation: 21    Number: 1197    Enthalpy:    90.9119    Mat-file: /home/USPEX/11/results1/USPEX.mat
Generation: 27    Number: 1542    Enthalpy:    90.9119    Mat-file: /home/USPEX/12/results1/USPEX.mat
Generation: 44    Number: 2519    Enthalpy:    90.9119    Mat-file: /home/USPEX/13/results1/USPEX.mat
Generation: 32    Number: 1821    Enthalpy:    90.9119    Mat-file: /home/USPEX/14/results1/USPEX.mat
Generation: 15    Number:  835    Enthalpy:    90.9119    Mat-file: /home/USPEX/15/results1/USPEX.mat
Generation: 43    Number: 2477    Enthalpy:    90.9119    Mat-file: /home/USPEX/16/results1/USPEX.mat
Generation: 40    Number: 2278    Enthalpy:    90.9119    Mat-file: /home/USPEX/17/results1/USPEX.mat
Generation: 24    Number: 1358    Enthalpy:    90.9119    Mat-file: /home/USPEX/18/results1/USPEX.mat
Generation: 14    Number:  757    Enthalpy:    90.9119    Mat-file: /home/USPEX/19/results1/USPEX.mat
Generation: 27    Number: 1532    Enthalpy:    90.9119    Mat-file: /home/USPEX/20/results1/USPEX.mat

Found structures numbers : 1326 1224 3451 1739  956 2055 1987 1241 1002 1641 1197 1542 2519 1821  835 2477 2278 1358  757 1532
Found generations numbers:   23   22   60   30   17   36   35   22   18   29   21   27   44   32   15   43   40   24   14   27

Success rate: 100 percent
Average number of generations to get E=90.912: 29
Average number of structures  to get E=90.912: 1647
Standard deviation: 670
```

▷ *variable* `fixRndSeed`

Meaning: The random seed number for USPEX calculations. For the same random seed, USPEX should produce the same result.

Default: `0`

Format:

```
-2000 : fixRndSeed
```

▷ *variable* `collectForces`

Meaning: Enables gathering all relaxation information in USPEX calculation, including total energies, forces on atoms, atomic positions, lattice parameters and stress tensors during structure relaxations. The information is stored in `FORCE.mat` file. Supported only for VASP, and is useful for machine learning.

Default: `0`

Format:

```
1 : collectForces
```

4.13 Seldom used keywords

▷ *variable* `mutationRate`

Meaning: Standard deviation of the strain matrix components for lattice mutation. The strain matrix components are selected randomly from the Gaussian distribution and are only allowed to take values between -1 and 1. Lattice mutation essentially incorporates the ideas of metadynamics into our method[6;24], where new structures are found by building up cell distortions of some known structure. Unlike in metadynamics, the distortions are not accumulated in our method, so the strain components should be large enough to obtain new structures.

Default: `0.5`

Format:

```
0.5 : mutationRate
```

It is a good idea to combine lattice mutation with a weak softmutation:

▷ *variable* `mutationDegree`

Meaning: The maximum displacement in softmutation in Å. The displacement vectors for softmutation or coordinate mutation are scaled so that the largest displacement magnitude equals `mutationDegree`.

Default: `3`×(average atomic radius)

Format:

```
2.5 : mutationDegree
```

▷ *variable* `ordering_active`

Meaning: Switch on the biasing of variation operators by local order parameters.

Default: `1`

Format:

```
1 : ordering_active
```

▷ *variable* `symmetrize`

Meaning: Switches on a transformation of all structures to standard symmetry-adapted crystallographic settings.

Default: `0`

Format:

```
1 : symmetrize
```

▷ *variable* `valenceElectr`

Meaning: Number of valence electrons for each type of atoms.

Default: these numbers are constants for all atoms, and we have tabulated them, no need to specify explicitly.

Format:

```
% valenceElectr
2 6
% EndValenceElectr
```

▷ *variable* `percSliceShift`

Meaning: Probability of shifting slabs (used in heredity) in all dimensions, 1.0 means 100%.

Default: `1.0`

Format:

```
0.5 : percSliceShift
```

▷ *variable* `maxDistHeredity`

Meaning: Specifies the maximal fingerprint cosine distances between structures that participate in heredity. This specifies the radius on the landscape within which structures can mate. Use with care (or do not use at all).

Default: `0.5`

Format:

```
0.5 : maxDistHeredity
```

▷ *variable* `manyParents`

Meaning: Specifies whether more than two slices (or more than two parent structures) should be used for heredity. This may be beneficial for very large systems.

Possible values (integer):

0 — only 2 parents are used, 1 slice each.

1 — many structures are used as parents, 1 slice each.

2 — two structures are used as parents, many slices (determined dynamically using parameters `minSlice` and `maxSlice`) are chosen independently from each one.

3 — two structures are used as parents, many slices (determined dynamically using parameters `minSlice` and `maxSlice`) are cut from the cell with a fixed offset. This is the preferred option for large systems. For example, we cut both structures into slices of approximately the same thickness and then choose the even slices from parent 1 and odd slices form parent 2, making a multilayered "sandwich", or a "zebra".

Default: `0`

Format:

```
3 : manyParents
```

`minSlice, maxSlice`: Determines the minimal and maximal thickness of the slices in Å that will be cut out of the parent structures to participate in creation of the child structure. We want the slices to be thick enough to carry some information about the parent (but not too thick to make heredity ineffective). Reasonable values for these parameters are around 1 and 6 Å, respectively.

For clusters, you can directly specify the number of parents participating in heredity (but we found this to be of little use):

▷ *variable* `numberparents`

Meaning: Defines the number of parents in heredity for clusters.

Default: `2`

Format:

```
2 : numberparents
```

5 Additional input for special cases

5.1 Variable-composition seraches: predicting novel compunds and their structures

To switch on the variable-composition mode, you have to:

1. Specify `301` or `311` or `201 : calculationType`.
2. Specify compositional building blocks in `numSpecies` (see the description of `numSpecies` variable).
3. Optionally specify the approximate atomic volumes for each atom type (or for each compositional block) using keyblock `Latticevalues`. However, we recommend relying on their default values, built into the program.
4. Specify the following options that are applicable only for variable-composition runs::

▷ *variable* `firstGeneMax`

Meaning: How many different compositions are sampled in the first generation. If 0, then the number is equal to `initialPopSize`/4. For binaries, we recommend `firstGeneMax`=11, for ternaries a higher value is needed, *e.g.* 30.

Default: `11`

Format:

```
10 : firstGeneMax
```

▷ *variable* `minAt`

Meaning: Minimum number of atoms (for `calculationType`=301/201/300) or molecules (for `calculationType`=311) in the primitive unit cell for the first generation.

Default: `No default`

Format:

```
10 : minAt
```

▷ *variable* `maxAt`

Meaning: Maximum number of atoms (for `calculationType`=301/201/300 or in META calculations) or molecules (for `calculationType`=311) in the primitive unit cell for the first generation. **Note:** minAt and maxAt should not differ by more than 2-3 times.

Default: `No default`

Format:

```
20 : maxAt
```

▷ *variable* fracTrans

Meaning: Percentage of structures obtained by transmutation. In this operator, a randomly selected atom is transmuted into another chemical

species present in the system — the new chemical identity is chosen randomly by default, or you can specify it in the block `specificTrans`, just like with specific permutation swaps.

Default: `0.1`

Format:

```
0.1 : fracTrans
```

▷ *variable* howManyTrans

Meaning: Maximum percentage of atoms in the structure that are being transmuted (0.1 = 10%). The fraction of atoms that will be transmuted is drawn randomly from a homogeneous distribution bounded from 0 to the fractional parameter `howManyTrans`.

Default: `0.2`

Format:

```
0.2 : howManyTrans
```

▷ *variable* specificTrans

Meaning: Specifies allowed transmutations.

Default: blank line (no specific transmutations)

Format:

```
% specificTrans
1 2
% EndTransSpecific
```

Note: In this case, atoms of type 1 could be transmuted into atoms of type 2 and vice versa. If you want to try all possible transmutations, just leave a blank line inside this keyblock.

In the case of variable-composition runs, parameter `keepBestHM` takes a new meaning — all structures on the convex hull (*i.e.*, thermodynamically stable states of the multicomponent system) survive, along with a few metastable states closest to the convex hull — the total number is `keepBestHM`. Also, parameter `stopCrit` has a new meaning in variable-composition calculations: its value (we recommend to set `stopCrit`=20 in variable-composition runs) indicates the maximum number of generations during which a structure can appear as a parent of new structures.

For variable-composition runs, it is particularly important to set up the first generation wisely. Choose a suitably large initial population size `initialPopSize`. Choose a reasonably large number of different compositions `firstGeneMax` to be sampled in the first generation (but not too large — each composition needs to be sampled several times at least). Finally, `minAt` and `maxAt` should not differ by more than 2 times, and you may need a few calculations with different system sizes: *e.g.*, 4–8, 8–16, 16–30 atoms, *etc.*

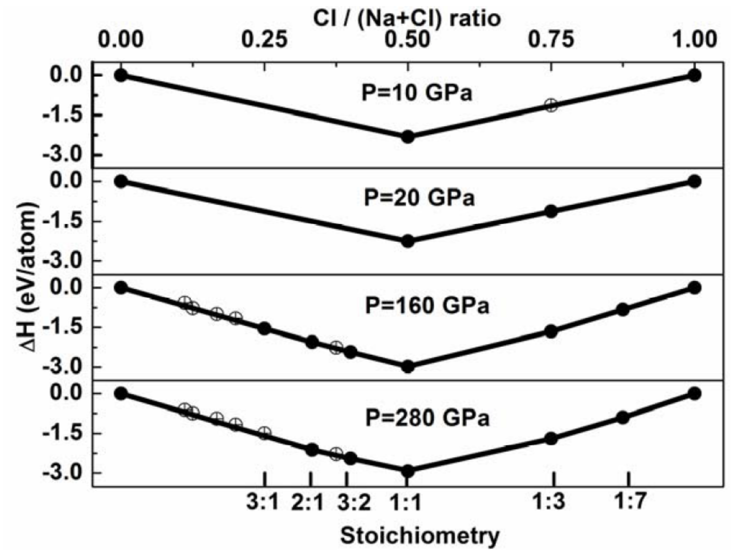


Figure 9: **Convex hull diagram for Na-Cl system at selected pressures.** Solid circles represent stable compounds; open circles — metastable compounds.

An additional comment for VASP users — if you want to perform a variable-composition run, let's say for the Na-Cl system, you should make sure the atomic types are given correctly in `INPUT.txt`, and put pseudopotential files `POTCAR_Na` and `POTCAR_Cl` in the folder ~`/StructurePrediction/Specific/`. USPEX will then recognize each atom and take each atom's `POTCAR` file appropriately for the calculations. Convex hull in Fig. 9 show stable sodium chlorides discovered using USPEX and confirmed by experiment[25].

5.2 Optimization of thermoelectric properties

Finding materials with high thermoelectric efficiency is an important problem. Therefore, it is necessary to enable the optimization of it.

The figure of merit (ZT) measures the thermoelectric efficiency of a material, and it is defined as

$$ZT = \frac{\sigma S^2 T}{\kappa_{\mathrm{E}} + \kappa_{\mathrm{PH}}}, \tag{7}$$

where σ is the electrical conductivity, S is the Seebeck coefficient, T is temperature, and κ_{E} and κ_{PH} are electronic and phonon parts of thermal conductivity, respectively. These values (except for κ_{PH}) can be calculated, within a constant relaxation time approximation, using the code `BoltzTraP`[26] on fully relaxed structures, from the electronic structure obtained by `VASP`. We strongly suggest to do Pareto optimization of both ZT and stability.

5.2.1 Installation of supporting software

To run optimization of thermoelectric figure of merit ZT using USPEX, some additional software is required.

1. Source code of `BoltzTraP`[26]. It can be downloaded from `http://www.icams.de/content/departments/cmat/boltztrap/`. Please, follow the manual attached to its source distribution for installation.
2. Python and python libraries. The interface of USPEX, `VASP` and `BoltzTraP`[26] has been tested on the Anaconda Python 2.7.10, which can be downloaded from `https://www.continuum.io/downloads`. In addition to the standard libraries, the following are also required:
    - `numpy`, version 1.4.1 or later.
    - `scipy`, version 0.11.0 or later.
    - `ase`, version 3.8.1 or later.
    - `pyspglib` (optional)
    - `matplotlib` (optional)

   With Anaconda Python, there is a large volume of stocked libraries that can be installed with the command `conda`. The following command lines are an example for installing libraries required for running a thermoelectric optimization.

   ```
   $ conda install numpy scipy matplotlib
   $ pip install ase pyspglib
   ```

   Under a normal Python distribution with `pip` installed, one can install required packages by typing the following command line.

   ```
   $ pip install numpy scipy matplotlib ase pyspglib
   ```

5.2.2 Specifying optimization of thermoelectric efficiency in `INPUT.txt`

For a thermoelectric efficiency optimization, the flag `optType` should be set to 14 as:

```
% optType
14
% EndOptType
```

Since the calculation of thermoelectric properties is performed on relaxed structures, a series of well-designed relaxation steps is necessary. Currently, the only implemented way to do this within USPEX is by combining `VASP` and `BoltzTraP`[26]. This is specified as follows:

```
% abinitioCode
1 1 1 (1 14)
% ENDabinit

% KresolStart
0.12 0.10 0.07 0.05 0.05
% Kresolend
```

In the above example, structure relaxation is performed by a series of VASP executions (four in total) with finer and finer **k**-point grid. Note that the last value of `KresolStart` in this example is not used, it is given only as a placeholder.

Optimization of thermoelectric efficiency is enabled only for `calculationType` 300 and 301. The following flags are also expected to be set. For a deeper understanding of them, the user is referred to the original BoltzTrap paper[26].

```
800.0     : BoltzTraP_T_max
50.0      : BoltzTraP_T_delta
0.15      : BoltzTraP_efcut
300.0     : TE_T_interest
0.45      : TE_threshold
ZT        : TE_goal
```

▷ *variable* `BoltzTraP_T_max`

Meaning: Maximum temperature (K) for calculations by BoltzTraP.

Default: `800.0`

Format:

```
     800.0 :  BoltzTraP_T_max
```

▷ *variable* `BoltzTraP_T_delta`

Meaning: Temperature (K) increments used by BoltzTraP.

Default: `50`

Format:

```
50.0 :  BoltzTraP_T_delta
```

▷ *variable* BoltzTraP_T_efcut

Meaning: This variable specifies the interval of the chemical potential μ (eV) for BoltzTraP calculations.

Default: `0.15`

Format:

```
0.2 :  BoltzTraP_efcut
```

▷ *variable* TE_T_interest

Meaning: Target temperature (K) at which you want to optimize thermoelectric efficiency.

Default: `300.0`

Format:

```
300.0 :  TE_T_interest
```

▷ *variable* TE_threshold

Meaning: This flag is used for reducing the amount of storage space required by the optimization and it sets a lower limit for ZT. Saving all information generated by BoltzTraP for a large number of structures could occupy a vast amount of disk space. Structures with a ZT value below `TE_threshold` are discarded.

Default: `0.5`

Format:

```
0.45 :  TE_threshold
```

▷ *variable* TE_goal

Meaning: The component of ZT to be optimized. Since the quantities defining ZT are tensorial, the user is expected to select a component of ZT to carry out the optimization. The options currently supported are the orientation-averaged ZT, or the diagonal components ZT_xx, ZT_yy, and ZT_zz.

Default: `ZT`

Format:

```
ZT : TE_goal
```

5.2.3 Output

The output of a thermoelectric efficiency optimization can be found in the sub-folder `TEproperties` under the appropriate `results-folder`. The maxima of ZT and their corresponding μ are listed in the summary file `summary.txt`. The first column is the ID of the structure. With the ID, its structure can be found in the file `gatheredPOSCARS`. Structures created by the `keepBest` operation are not listed in the summary file, as each of them is a copy of another structure listed. A sample part of the summary file is as follows:

```
#                 trace
# ID   mu_max_1  ZT_max_1  mu_max_2  ZT_max_2
    1  0.019900  0.314997 -0.186600  0.294976
    2  0.152270  0.913590  0.002270  0.864951
    3 -0.021300  4.951155  0.152200  3.244481
    4 -0.194630  0.677683  0.054370  0.628948
    6  0.024120  6.144226  0.053620  0.914606
    7  0.030460  0.887808  0.041960  0.831734
    9  0.167860  0.479234  0.159360  0.446431
   10  0.133260  0.893694  0.144760  0.822560
   12  0.063250  0.903193  0.075750  0.843479
   13  0.044820  1.793899  0.004820  1.026578
```

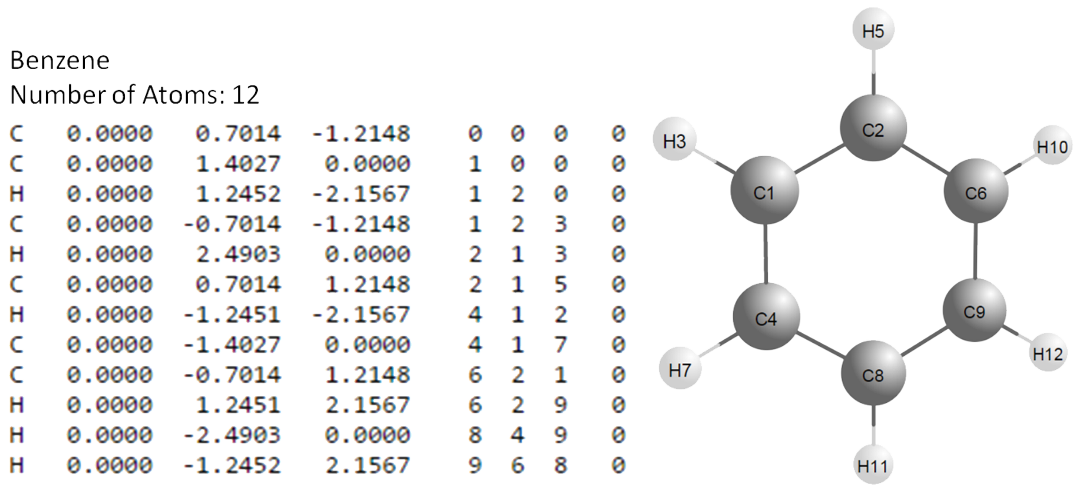
Individual thermoelectric properties (such as σ, S, κ_{E} and power factor σS^2) can also be found in the folder `TEproperties` with corresponding structure ID number appended in the file name.

Note that, due to numerical instabilities in the computation of thermoelectric properties, some of the results might be incorrect, in which case the values of ZT can be tens or hundreds of thousands times a normal value. USPEX makes sure that such values are discarded. To reduce frequency of occurence of such numerical instabilities, we recommend to symmetrize relaxed structures when their band structures (necessary for ZT calculations) are computed.

5.3 Molecular crystals and polymers

5.3.1 Molecular crystals, `calculationType`=310/311

For a molecular crystal, the `MOL_1` file describes the internal geometry of the molecule from which the structure is built. The `Z_Matrix` file is created using the information given in the `MOL_1` file, *i.e.*, bond lengths and all necessary angles are calculated from the Cartesian coordinates. The lengths and angles that are important should be used for the creation of `Z_Matrix` — this is exactly what columns 5–7 specify. Let's look at the `MOL_1` file for benzene C_6H_6:



```
Benzene
Number of Atoms: 12
C   0.0000   0.7014  -1.2148    0  0  0   0
C   0.0000   1.4027   0.0000    1  0  0   0
H   0.0000   1.2452  -2.1567    1  2  0   0
C   0.0000  -0.7014  -1.2148    1  2  3   0
H   0.0000   2.4903   0.0000    2  1  3   0
C   0.0000   0.7014   1.2148    2  1  5   0
H   0.0000  -1.2451  -2.1567    4  1  2   0
C   0.0000  -1.4027   0.0000    4  1  7   0
C   0.0000  -0.7014   1.2148    6  2  1   0
H   0.0000   1.2451   2.1567    6  2  9   0
H   0.0000  -2.4903   0.0000    8  4  9   0
H   0.0000  -1.2452   2.1567    9  6  8   0
```

Figure 10: **Sample of `MOL_1` file and illustration of the corresponding molecular structure.**

The 1st atom is H, its coordinates are defined without reference to other atoms ("0 0 0").

The 2nd atom is C, its coordinates (in molecular coordinate frame) in `Z_matrix` will be set only by its distance from the 1st atom (*i.e.* H described above), but no angles — ("1 0 0").

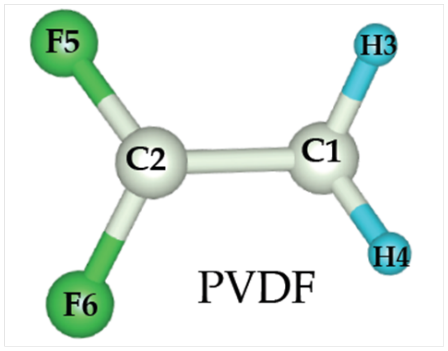
The 3rd atom is C, its coordinates will be set by its distance from the 2nd atom, and the bond angle 3-2-1, but not by torsion angle — hence we use "2 1 0".

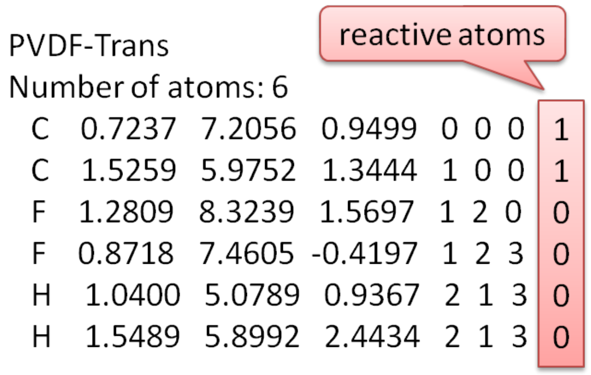
The 4th atom is C, its coordinates will be set by its distance from the 3rd atom, bond angle 4-3-2, and torsion angle 4-3-2-1 — hence, we use "3 2 1" and so forth... until we reach the final, 12th atom, which is H, defined by its distance from the 7th atom (C), bond angle 12-7-6 and torsion angle 12-7-6-11 — hence "7-6-11".

The final column is the flexibility flag for the torsion angle. For example, in C4, the tosion angle is defined by 4-3-2-1. This flag should be 1 for the first three atoms, and 0 — for the others, if the molecule is rigid. If any other flexible torsion angle exists, specify 1 for this column.

5.3.2 Polymeric crystals, `calculationType`=110 ("linear chain model")

For polymers, the `MOL_1` file is used to represent the geometry of a monomeric unit, in the same style as for molecular crystals, except that we use the last column to specify the reactive atoms as shown in the `MOL_1` file for PVDF:




reactive atoms

```
PVDF-Trans
Number of atoms: 6
  C   0.7237  7.2056   0.9499  0 0 0  1
  C   1.5259  5.9752   1.3444  1 0 0  1
  F   1.2809  8.3239   1.5697  1 2 0  0
  F   0.8718  7.4605  -0.4197  1 2 3  0
  H   1.0400  5.0789   0.9367  2 1 3  0
  H   1.5489  5.8992   2.4434  2 1 3  0
```

Figure 11: **Sample of MOL_1 file of PVDF and illustration of the corresponding monomeric structure.**

5.3.3 Additional inputs for classical forcefields

Above we described a generic MOL_1 file format. However, some classical forcefield based codes need additional information. For instance, GULP needs to specify the chemical labels and charge. The MOL_1 file for aspirin can be written in the following way:

```
Aspirin_charge
Number of atoms: 21
H_1      0.2310     3.5173     4.8778     0   0   0    1      0.412884
O_R      0.7821     4.3219     4.9649     1   0   0    1     -0.676228
C_R      0.4427     5.0883     6.0081     2   1   0    1      0.558537
O_2     -0.5272     4.5691     6.6020     3   2   1    0     -0.658770
C_R      1.0228     6.3146     6.3896     3   2   4    0      0.116677
C_R      2.1330     6.8588     5.6931     5   3   2    0      0.311483
C_R      0.4810     7.0546     7.4740     5   3   6    0     -0.119320
O_R      2.8023     6.2292     4.6938     6   5   3    0     -0.574557
C_R      2.6211     8.1356     6.0277     6   5   8    0     -0.083091
C_R      0.9966     8.3146     7.8237     7   5   3    0     -0.103442
H_2     -0.3083     6.6848     8.0128     7   5  10    0      0.198534
C_R      3.6352     5.1872     4.9079     8   6   5    0      0.609295
C_R      2.0623     8.8613     7.0940     9   6   5    0     -0.119297
H_2      3.3963     8.5283     5.4906     9   6  13    0      0.174332
H_2      0.5866     8.8412     8.6013    10   7  13    0      0.205960
O_2      3.9094     4.7941     6.0632    12   8   6    0     -0.588433
C_3      4.2281     4.5327     3.7638    12   8  16    0     -0.271542
H_2      2.4227     9.7890     7.3367    13   9  10    0      0.196738
H_2      3.4269     4.1906     3.1183    17  12   8    0      0.151315
H_2      4.8283     3.6848     4.0792    17  12  19    0      0.131198
H_2      4.8498     5.2464     3.2337    17  12  19    0      0.127726
```

Here, the keyword `charge` in the title tells the program to read the charge in the additional (last) column.

To work with Tinker, the additional column must specify the atomic type label as follows:

```
Urea
Number of atoms: 8
C     0.000000    0.000000    0.000000   0 0 0 1   189
O     0.000000    0.000000    1.214915   1 0 0 1   190
N     1.137403    0.000000   -0.685090   1 2 0 1   191
N    -1.137403    0.000000   -0.685090   1 2 3 0   191
H     1.194247    0.000000   -1.683663   4 1 3 0   192
H    -1.194247    0.000000   -1.683663   4 1 3 0   192
H     1.998063    0.000000   -0.138116   2 1 3 0   192
H    -1.998063    0.000000   -0.138116   2 1 3 0   192
```

5.3.4 How to prepare the `MOL` files

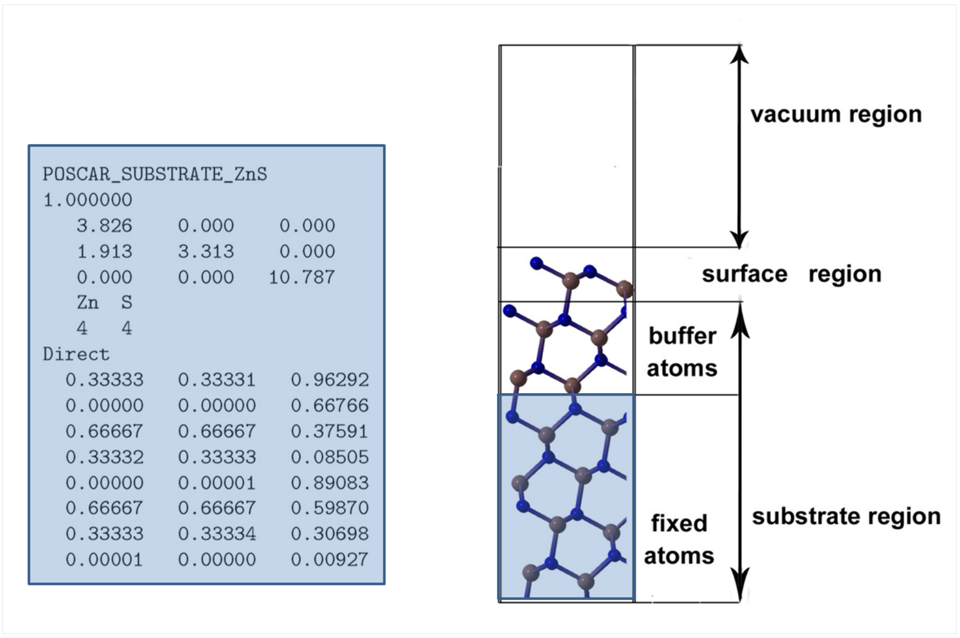
There are plenty of programs which can generate Zmatrix style files, such as Molden, Avogadro, and so on. Experienced users might have their own way to prepare these files. For the users' convenience, we have created an online utility to allow one to generate the USPEX-style `MOL` file just from a file in XYZ format. Please try this utility at http://uspex-team.org/online_utilities/zmatrix/.

5.4 Surfaces

To predict surface reconstructions, you have to:

- Specify `200` or `201 : calculationType`.
- Provide a file containing substrate in VASP5 POSCAR format, as shown on Fig. 12. This can be done using online utility: `http://uspex-team.org/online_utilities/substrate/` (after you get your slab model, you need to put it in your folder and run USPEX one time — it will generate for you the "final" `POSCAR_SUBSTRATE` file at stop. With this file you can now run your USPEX calculations).
- Specify the following parameters:

```
% symmetries
2-17
% endSymmetries
```



```
POSCAR_SUBSTRATE_ZnS
1.000000
   3.826    0.000    0.000
   1.913    3.313    0.000
   0.000    0.000   10.787
   Zn  S
   4   4
Direct
  0.33333   0.33331   0.96292
  0.00000   0.00000   0.66766
  0.66667   0.66667   0.37591
  0.33332   0.33333   0.08505
  0.00000   0.00001   0.89083
  0.66667   0.66667   0.59870
  0.33333   0.33334   0.30698
  0.00001   0.00000   0.00927
```

Figure 12: **Surface model used in USPEX.** Note that POSCAR_SUBSTRATE shall exactly represent the geometrical information of its bulk crystal without vacuum. If the input has large vacuum region, the program will automatically delete it and provide a new file called POSCAR_SUBSTRATE_NEW, and this file has to be used in the calculation (renamed as POSCAR_SUBSTRATE).

Note: If the `symmetries` tag is present, USPEX will try to generate surface structures using plane groups.

▷ *variable* `thicknessS`

Meaning: Thickness of surface region. Adatoms are allowed only in this region.

Default: `2.0` Å

Format:

```
3.5 : thicknessS
```

▷ *variable* `thicknessB`

Meaning: Thickness of buffer region in substrate. This region is part of POSCAR_SUBSTRATE, and is allowed to relax.

Default: `3.0` Å

Format:

```
3.0 : thicknessB
```

▷ *variable* `reconstruct`

Meaning: Maximum multiplications of the surface cell, to allow for complex reconstructions.

Default: `1`

Format:

```
    1 :  reconstruct
```

USPEX is capable of predicting surface reconstructions with variable number of surface atoms. In this case, stable surface reconstructions are dictated by chemical potentials[27]. A typical set of input is the following:

```
******************************************
*      TYPE OF RUN AND SYSTEM            *
******************************************
USPEX  : calculationMethod (USPEX, VCNEB, META)
201    : calculationType (dimension: 0-3; molecule: 0/1; varcomp: 0/1)

% atomType
Si O
% EndAtomType

% numSpecies
2 4
% EndNumSpecies
```

Here we specify the maximum number of surface atoms in a 1×1 cell.

```
******************************************
*              SURFACES                  *
******************************************
% symmetries
2-17
% endSymmetries

% StoichiometryStart
1 2
% StoichiometryEnd
```

This defines the stoichiometry of the bulk.

```
-23.7563 : E_AB (DFT energy of AmBn compound, in eV per formula unit)
-5.4254  : Mu_A (DFT energy of elemental A, in eV/atom)
```

```
-4.9300  : Mu_B (DFT energy of elemental B, in eV/atom)

3.5      : thicknessS (thickness of surface region, 2 A by default )
3.0      : thicknessB (thickness of buffer region in substrate, 3 A by default)
4        : reconstruct (maximum multiplications of cell)
```

At the moment, USPEX supports variable-composition calculation for the following cases:

- Reconstructions of elemental surfaces (such as C@diamond(100) surface).
- Reconstructions of surfaces at binary compounds (such as GaN(0001) surface).
- Reconstructions involving foreign species on elemental surfaces (such as PdO@Pd(100) surface).

5.5 Two dimensional (2D) crystals

5.5.1 2D-crystals, calculationType = -200/-201

For 2D-crystals, USPEX allows fixed and variable-composition calculations (-200/-201). Elemental and binary systems can be studied for now, ternary and more complex systems will be enabled later. Different from surface calculation (200/201), we need neither reference energies nor the substrate, but only the thickness confinement of initial 2D structures, for example:

```
3.0 : thicknessS
```

It means that we produce the initial 2D structures within a 3.0 Angstrom thick slab. The maximum final structures thickness is empirically confined as thicknessS + 3. **Importantly**, we also need to give a large enough vacuum size for 2D-crystals (default value is 10 Angstrom, sometimes this is enough). Different from surfaces (where symmetry is described by 17 plane groups), symmetry of 2D-crystals is described by 80 layer groups, which are listed in Appendix 9.5.

```
% symmetries
2-80
% endSymmetries
```

5.5.2 Fixed-composition structure prediction

We define the stoichiometric ratio by `numSpecies` keyblock, and then specify the minimum and maximum total numbers of atoms in the unit cell by `minAt` and `maxAt`. For example, here is what you would specify to perform 2D-crystal search for B_2O_3 system with 4-20 atoms in the cell, that is, we are considering B_2O_3, B_4O_6, B_6O_9, B_8O_{12} within one USPEX calculation:

```
% numSpecies
2 3
% EndNumSpecies

% atomType
B O
% EndAtomType

4    : minAt
20   : maxAt
```

5.5.3 Variable-composition structure prediction

For a full variable-composition study of 2D-crystals, similar with 3D cases, we set up the matrix `numSpecies`:

```
% numSpecies
1 0
0 1
% EndNumSpecies
```

Note: Because we have confined the thickness for final 2D structures, we discard the structures with thickness after relaxation greater than `thicknessS`+3. For variable-composition calculations, we may need a relatively large value of `initialPopSize` (we suggest twice the `populationSize`), for example:

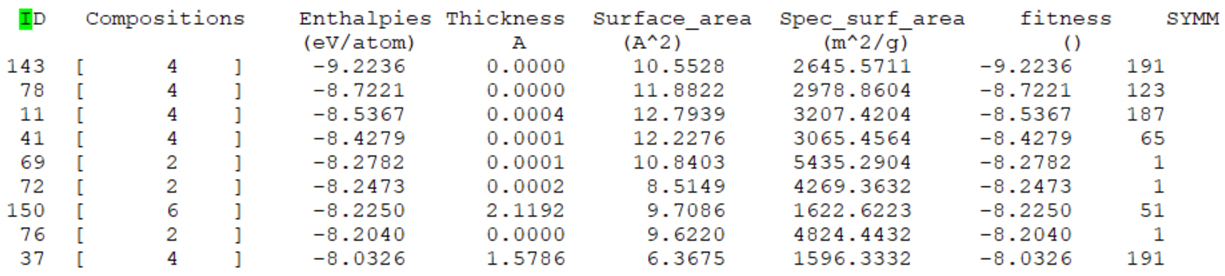
```
 90      : populationSize
180      : initialPopSize
100      : numGenerations
```

5.5.4 Summary of good structures

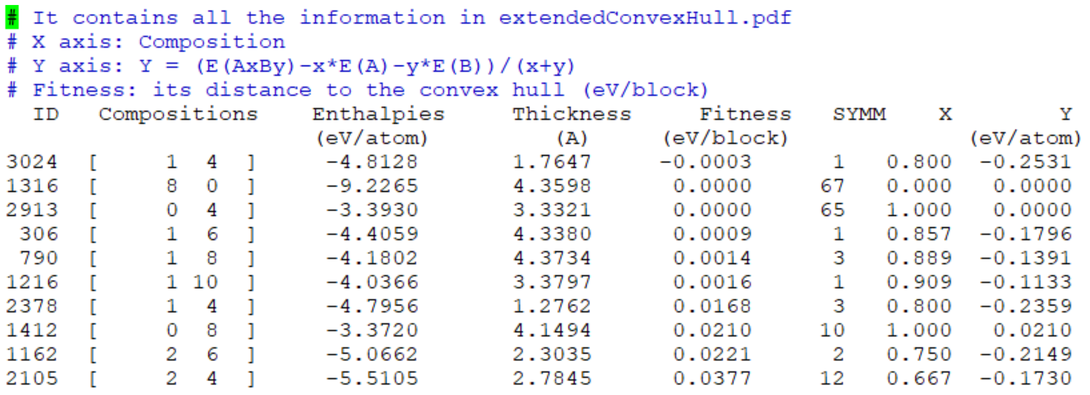
For -200, the information about good 2D structures can be found in `goodStructures`, which looks as shown below: Fig. **??** shows an example output.

In the -201 regime, the convex hull graph is shown in the file `extendedConvexHull.pdf`, and information about all structures (starting from the stable ones) is output in the `extended_convex_hull` file, which looks like this:

We provide an example (EX30) of a variable-composition prediction of stable 2D-crystals in the Sn-S system.



```
ID    Compositions      Enthalpies Thickness  Surface_area  Spec_surf_area     fitness     SYMM
                         (eV/atom)      A        (A^2)         (m^2/g)           ()
143   [        4     ]    -9.2236     0.0000     10.5528      2645.5711      -9.2236     191
 78   [        4     ]    -8.7221     0.0000     11.8822      2978.8604      -8.7221     123
 11   [        4     ]    -8.5367     0.0004     12.7939      3207.4204      -8.5367     187
 41   [        4     ]    -8.4279     0.0001     12.2276      3065.4564      -8.4279      65
 69   [        2     ]    -8.2782     0.0001     10.8403      5435.2904      -8.2782       1
 72   [        2     ]    -8.2473     0.0002      8.5149      4269.3632      -8.2473       1
150   [        6     ]    -8.2250     2.1192      9.7086      1622.6223      -8.2250      51
 76   [        2     ]    -8.2040     0.0000      9.6220      4824.4432      -8.2040       1
 37   [        4     ]    -8.0326     1.5786      6.3675      1596.3332      -8.0326     191
```



```
# It contains all the information in extendedConvexHull.pdf
# X axis: Composition
# Y axis: Y = (E(AxBy)-x*E(A)-y*E(B))/(x+y)
# Fitness: its distance to the convex hull (eV/block)
  ID    Compositions      Enthalpies      Thickness      Fitness     SYMM    X          Y
                           (eV/atom)         (A)        (eV/block)                (eV/atom)
3024   [     1  4  ]       -4.8128        1.7647       -0.0003        1   0.800   -0.2531
1316   [     8  0  ]       -9.2265        4.3598        0.0000       67   0.000    0.0000
2913   [     0  4  ]       -3.3930        3.3321        0.0000       65   1.000    0.0000
 306   [     1  6  ]       -4.4059        4.3380        0.0009        1   0.857   -0.1796
 790   [     1  8  ]       -4.1802        4.3734        0.0014        3   0.889   -0.1391
1216   [     1 10  ]       -4.0366        3.3797        0.0016        1   0.909   -0.1133
2378   [     1  4  ]       -4.7956        1.2762        0.0168        3   0.800   -0.2359
1412   [     0  8  ]       -3.3720        4.1494        0.0210       10   1.000    0.0210
1162   [     2  6  ]       -5.0662        2.3035        0.0221        2   0.750   -0.2149
2105   [     2  4  ]       -5.5105        2.7845        0.0377       12   0.667   -0.1730
```

5.6 Clusters

To predict structures of clusters, you have to:

- Specify `000 : calculationType`.
- Specify the following parameters:

5.7 Evolutionary metadynamics code

This is a very powerful method for finding the global minimum, as well as many low-energy metastable structures that are potentially kinetically accessible from the starting structure. The starting structure has to be high-quality and is given in the file `POSCAR_1`.

Evolutionary metadynamics is only enabled with the VASP and GULP codes at the moment.

To switch on the evolutionary metadynamics mode, you have to:

1. Specify

   `META : calculationMethod`

   `300 : calculationType`

2. Create file `POSCAR_1` in the VASP5 format in your folder (evolutionary metadynamics requires a good starting structure, relaxed at the pressure of interest).
3. Specify the population size (in this case, this is the number of softmutations at each metastep):

   ```
   30 :  populationSize
   ```

4. Specify the pressure:

   ▷ *variable* `ExternalPressure`

   Meaning: The pressure at which you want to perform the calculation, in GPa.

   Default: no default

   Format:

   ```
       10 :  ExternalPressure (GPa)
   ```

5. Specify the following metadynamics-only options:

▷ *variable* `GaussianWidth`

Meaning: The width of each of the Gaussians added to the energy surface to accelerate phase transitions. A good rule of thumb is to choose a value close to 0.10–0.15L, where L is the minimum length of the unit cell, in Angstroms.

Default: $0.10 \times L$ (Å)

Format:

```
    0.80 :  GaussianWidth
```

▷ *variable* `GaussianHeight`

Meaning: The height of each of the Gaussians added to the energy surface to accelerate phase transitions. A good rule of thumb (Martoňák *et al.*, 2005) is to choose a value close to $L(\delta h)^2 G$, where L is the average length of the unit cell in Angstroms, δh is the Gaussian width in Angstroms (see below), and G is the shear modulus in kbars.

Default: $1000 \times (0.10 \times L)^2 \times L = 10 \times L^3$ (Å^{3}kbar)

Format:

```
    2000 :  GaussianHeight
```

▷ *variable* `FullRelax`

Meaning: Metadynamics as such only relaxes structures within a fixed cell. For analysis, you need to perform complete structure relaxation (*i.e.* relaxing also the cell).

- `FullRelax`=0 — no full relaxation will be performed (very fast option, but inconvenient for analysis of the results).
- `FullRelax`=1 — only the best structure of the generation will be fully relaxed (also fast, sometimes sufficient).
- `FullRelax`=2 — all inequivalent structures are fully relaxed (still fast, only ∼2 times slower than `FullRelax`=1, but provides a lot more insight. Strongly recommended for most cases).

Default: `2`

Format:

```
2 :  FullRelax
```

For full relaxation, when performing evolutionary metadynamics the format of the block `abinitioCode` is slightly different, for example:

```
abinitioCode
3 3 3 3 (3 3)
ENDabinit
```

In the example above, there are four stages of relaxation within a fixed cell, and two stages of full relaxation (in parentheses). Remember that in the last fixed-cell stage of relaxation, pressure tensor must be accurate — this is what drives metadynamics.

▷ *variable* `maxVectorLength`

Meaning: Together with `minVectorLength`, this sets boundary values for basic cell lengths in evolutionary metadynamics (note that this is a different meaning for `minVectorLength` from normal calculations, and `maxVectorLength` is only used in evolutionary metadynamics). When any of the basic cell lengths becomes smaller than `minVectorLength` or larger than `maxVectorLength`, we add a steep correction "force" in metadynamics, which drives cell evolution towards "good" values. The correction forces are exactly zero when all basic cell lengths are in the "good" range.

Default: `No default`

Format:

```
12.0 :  minVectorLength
```

Note: if you specify maxAt variable, a generalized evolutionary metadynamics calculation (exploiting different supercells) will be performed.

When you run metadynamics, additional files will be found in the `results1` folder, most importantly:

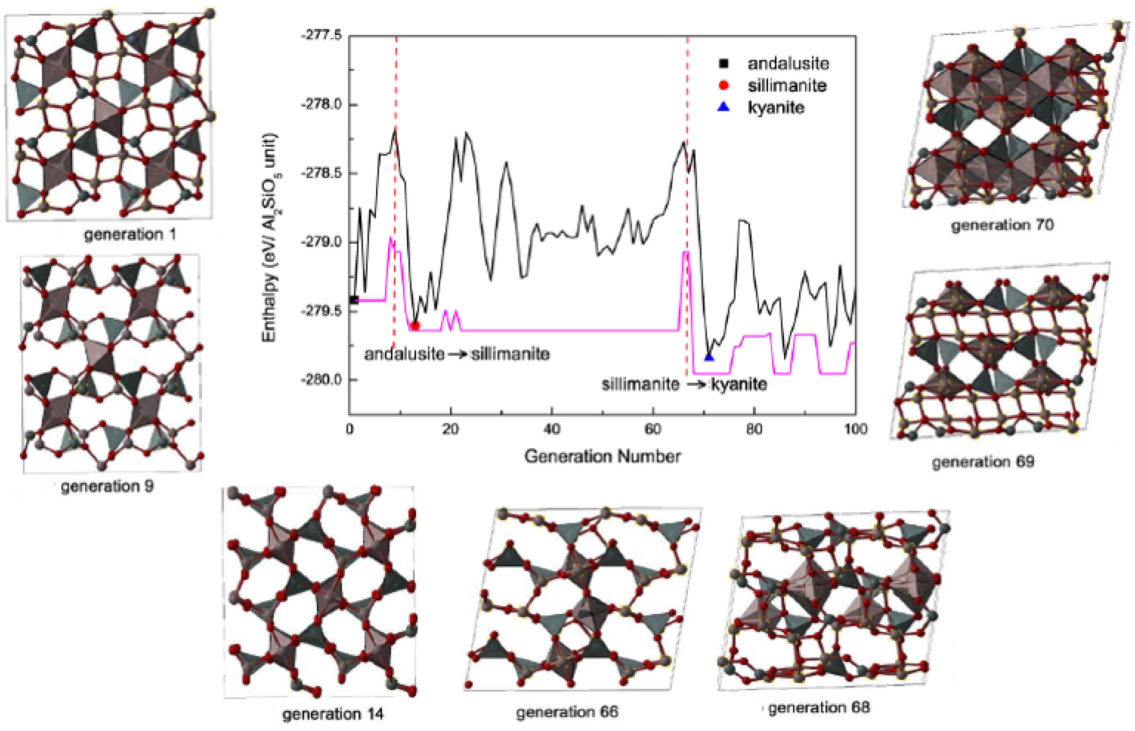


Figure 13: **Enthalpy evolution of Al_2SiO_5 starting with andalusite at 10 GPa (black line: enthalpies for best structures with constant h; magenta line: enthalpies for best structures after full relaxation).** Sequence of structures obtained in this run: generation 1 (andalusite) → generation 9 (sillimanite) → generation 14 → generation 66 → generation 68 → generation 69 → generation 70 (kyanite).

- `force.dat` — analysis of forces on the cell, internal (`f_c`) and from the Gaussians (`f_g`);
- `presten` — pressure tensor;
- `lattice.dat` — cell shape change during the simulation;
- `enthalpies` and `enthalpies_relaxed` — enthalpies of structures at each metastep at fixed cell and after full relaxation, respectively;
- `gatheredPOSCARS` and `gatheredPOSCARS_relaxed` — structures at fixed cell and after full relaxation, respectively.

Fig. 13 shows an example of use of evolutionary metadynamics: starting from one Al_2SiO_5 polymorph (andalusite), we obtained the other two known polymorphs (kyanite and sillimanite) and non-trivial phase transformation mechanisms.

5.8 Particle swarm optimization (PSO) code

In the field of crystal and cluster structure prediction, several approaches proved to be successful for small systems. Particle Swarm Optimization (PSO), pioneered in this field by Boldyrev[28], is a special class of evolutionary algorithms where a population (swarm) of candidate solutions (called "particles") is moved in the search space according to a few simple formulae. The movements of the particles are guided by their own best known position in the search space as well as the entire swarm's best known position. Initially, the coordinates χ and 'velocities' υ of the particles are generated randomly. Then at every step, the positions and velocities are updated according to the formulae:

$$\begin{aligned} \upsilon_i' &= \omega \cdot \upsilon_i + \varphi_p \cdot r_p \cdot (p_i - \chi_i) + \varphi_g \cdot r_g \cdot (g - \chi_i), \\ \chi_i' &= \chi_i + \upsilon_i'. \end{aligned} \tag{8}$$

Here ω, φ_p and φ_g are weight factors that control the behavior and efficiency of the PSO algorithm; r_p and r_g are random numbers in the [0; 1] range generated separately for every particle at every step; p_i is the best known position of particle i and g is the best known position of the entire swarm.

Such an algorithm, despite its simplicity, can work[28]. Key points to improve with respect to previous implementations[28;29] are (1) metric of the search space (it is not trivial to map crystal structures uniquely onto a coordinate system) and (2) ways to evolve structures in PSO, *i.e.* variation operators.

Evolving the particles by determining the speed υ_i (8) directly from coordinates of the atoms and cell parameters of two structures (as in Ref.[29]) cannot be productive. Our solution is to use fingerprint distances[17] as the most natural metric for the energy landscape, and variation operators of USPEX for evolving the 'PSO particles' (*i.e.* structures) as the

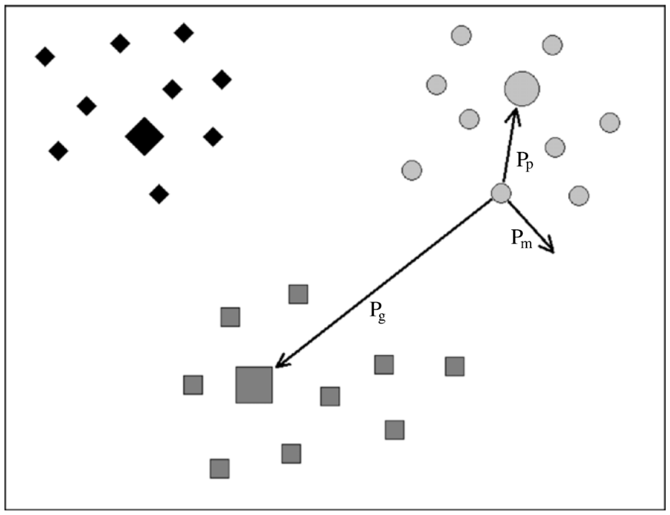


Figure 14: **Illustration of PSO-USPEX hybrid algorithm for the population of three individuals (marked by diamonds, squares and circles) after 10 generations.** Best position for each particle is marked by an enlarged symbol. The best structure is the big square. The structure shown by circle can be either mutated, create a child with its historically best position (large circle) or the best position of entire population (large square) using heredity operator with probabilities P_m, P_p and P_g, respectively.

most efficient unbiased ways to evolve a population of structures. Namely, the particle is either mutated (to imitate a random move), or participates in heredity with its best known position or in heredity with the best known population position (to imitate PSO moves in the direction of these positions). Instead of applying at each step all moves with some weights (see Eq. 8), we apply them one at a time with probabilities described by formulae:

$$\begin{gathered} P_m = \frac{\omega}{\Sigma}; \qquad P_p = \frac{\varphi_p \cdot r_p \cdot D_p}{\Sigma}; \qquad P_g = \frac{\varphi_g \cdot r_g \cdot D_g}{\Sigma}; \\ \Sigma = \omega + \varphi_p \cdot r_p \cdot D_p + \varphi_g \cdot r_g \cdot D_g, \end{gathered} \tag{9}$$

where D_p is a fingerprint distance between current and best position of a particle, while D_g is a fingerprint distance between the current position of the particle and best known position of the entire population. Our tests, performed on a few diverse systems, show that this approach (which we call "cor-PSO", *i.e.* corrected PSO) is relatively successful and works better than previous versions of PSO, but still cannot compete with the USPEX algorithm[2;30] for success rate or efficiency.

The following variables are unique for `calculationMethod`=PSO:

▷ *variable* `PSO_softMut`

Meaning: Weight of softmutation (ω in eq. 9).

Default: `1`

Format:

```
1 :  PSO_softMut
```

▷ *variable* `PSO_BestStruc`

Meaning: Weight of heredity with the best position of the same PSO particle (φ_p in eq. 9).

Default: `1`

Format:

```
1 :  PSO_BestStruc
```

▷ *variable* `PSO_BestEver`

Meaning: Weight of heredity with the globally best PSO particle (φ_g in eq. 9).

Default: `1`

Format:

```
1 :  PSO_BestEver
```

6 Prediction of phase transition pathways

Phase transitions determine many aspects of the behavior of materials. Thus, it is essential to reveal possible mechanisms of structural phase transitions.

6.1 Pmapaths: geometric method for predicting solid-solid phase transition mechanisms

This tool, written by V. Stevanovic, P. Graf and F. Therrien, and modified for our purposes by A. Samtsevich, solves the hard problem of finding the most conservative mapping between two given crystal structures. This intuitively corresponds to the easiest phase transformation mechanism(s). Pmpaths enumerates and ranks different pathways connecting two user-provided crystal structures. These pathways will then need to be optimized by the VCNEB method (described in the next section).

This algorithm first searches for mapping of the unit cells, by optimizing the "overlap" of the cell, and permuting the unit cell vectors so the coordinate system in the rotated frame that gives best overlap are as closely aligned as possible. This is the starting point for determining the best mappings between atomic positions in the two crystal structures, and subsequent analysis.

The code relies on some libraries:

```
munkres
scikit_learn
spglib
cython
```

All of them can be installed with **pip**

To run the code, try, for example

```
python3 run_pmpaths.py -t 4 -z traj_A2B -n 20 -A POSCAR_A -B POSCAR_B -b 3.6 --numtraj=5
```

A help menu is built in:

```
Usage: run_pmpaths.py [options]

Options:
  -h, --help            show this help message and exit
  -A A, --A=A           poscar 1
  -B B, --B=B            poscar 2
  -t OUTPUT_TILES, --tiles=OUTPUT_TILES
                        how many cells to tile in output
  -H, --hlst            perform HLST fitting
  -v VERBOSE, --verbose=VERBOSE
                        verbosity
  -z TRAJDIR, --trajdir=TRAJDIR
                        where to dump trajectory files
  -c MIN_CLUSTER_SIZE, --min_cluster_size=MIN_CLUSTER_SIZE
                        minimum size of atom clusters
  -s SHIFTING, --shifting=SHIFTING
                        control shift of inequiv atoms to origin (0:none,
                        1:one of each inequivalent subgroup,2:all)
  -b BOND_LEN, --bond_len=BOND_LEN
                        bond length
  -n FRAMES, --frames=FRAMES
                        how many frames in trajectory
  -y, --nocheck_syms    don't check syms
  -u, --nocheck_ucells  don't check more than one unit cell pairing, but DO do
                        gruberization
  -w, --use_given_ucells
                        use given unit cells as is, no gruberization
  -d, --noucell-dist    don't include unit cell vector movement in distance
                        measure
  -f, --get-fast        use bonding to specifically search for FAST
  -e TOL, --tol=TOL     tolerance for coordination calcs
  --numtraj=N          , where N is the number of pathways that will be in the output
```

The output of run_pmpaths.py (besides all it writes to stdout) is a directory ("traj_A2B") that contains **n** frames ("images") along the path. These are then further should be refine by the using VCNEB method implemented in USPEX.

6.2 Variable-cell nudged elastic band (VCNEB) method

Prediction of a phase transition mechanism can be considered as a double-ended problem, in which the algorithm has to locate the intermediate states. The nudged elastic band (NEB)[31;32;33] method is a widely used technique for solving double-ended problems, an efficient and robust approach for seeking reaction paths and saddle points along the "minimum energy path" (MEP) on the potential energy surface between the two endpoints. The NEB method has been successfully applied to study molecular chemical reactions, and defect migration, and in principle it could provide the energy barrier between the given initial and final states of a phase transition process. Unfortunately, most of the problems treated by the NEB method are considered under the constraint of constant unit cell — which precludes it from being used for phase transitions (which involve the variation of the unit cell along the transition path).

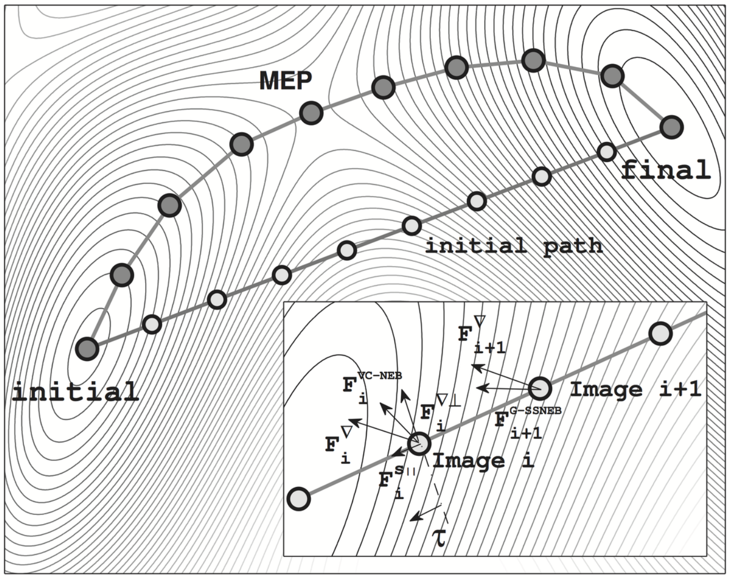


Figure 15: **The minimum energy path (line with gray circles) and initial path on a model 2D enthalpy surface.** The forces in the VCNEB method on image i are shown in the inset. $\mathbf{F}_i^{\nabla}$ is the potential force in the gradient direction. $\mathbf{F}_i^{\nabla\perp}$ and $\mathbf{F}_i^{s\parallel}$ are the transverse component of $\mathbf{F}_i^{\nabla}$ and the spring force, respectively.

The variable-cell NEB (VCNEB) method[14], which we have developed, treats the cell and atomic coordinates on an equal footing and operates in an expanded configuration space under the condition of constant pressure. Our VCNEB method framework has been added to the USPEX code in 2013. The VCNEB method is a more general tool for exploring the activation paths between the two endpoints of a phase transition process within a larger configuration space. Every structure on the pathway in the VCNEB method is regarded as an "Image".

6.2.1 Input options for VCNEB

The VCNEB method is only enabled with the VASP, GULP and Quantum Espresso codes at the moment.

To switch on the VCNEB mode, you have to:

1. Specify

   `VCNEB : calculationMethod`

2. Create a file `Images` in the VASP5 format in your folder (VCNEB requires at least two structures, initial and final phases, to run the phase transition pathway prediction).

3. Specify the following VCNEB options:

▷ *variable* `vcnebType`

Meaning: Specifies type of the VCNEB calculation. This variable consists of three indices: *calculation mode*, *image number variability*, and *spring constant variability*:

- calculation option:

  "1" — the VCNEB method;

  "2" — structure relaxation mode with no VCNEB calculation.

- `Variable-Image-Number` method:

  "0" — the number of images in VCNEB calculation is fixed;

  "1" — the number of images in VCNEB calculation is variable.

- variability of spring constant:

  "0" — fixed spring constant;

  "1" — variable spring constant.

Default: `110`

Format:

```
111 :  vcnebType
```

Note: If `vcnebType`=111, *i.e.*, a calculation for VCNEB calculation with variable number of Images and variable spring constant is to be performed. We strongly suggest users to run a variable number of images in VCNEB calculations when investigating reconstructive phase transitions.

▷ *variable* `numImages`

Meaning: Initial number of images to perform the calculation.

Default: `9`

Format:

```
13 :  numImages
```

▷ *variable* `numSteps`

Meaning: Maximum number of VCNEB steps.

Default: `600`

Format:

```
500 :  numSteps
```

Notes: (1) When `numSteps`=-1, the initial pathway will only be generated with no further optimization. (2) Convergence of VCNEB pathways is usually rather slow. We recommend to set `numSteps` to at least 500.

▷ *variable* `optReadImages`

Meaning: Options for reading the `Images` file:

- "0" — All images (`numImages`) are needed and specified in `Images` file;
- "1" — Only initial and final images are needed and would be read in `Images` file;
- "2" — The initial, final and any specified intermediate Images will be read in `Images` file.

Default: `2`

Format:

```
1 :  optReadImages
```

Note: In all options, the initial and final images must be specified. Automatic linear interpolation will be applied to generated the initial Images in option 1 and 2.

▷ *variable* `optimizerType`

Meaning: Optimization algorithm option of structure relaxation:

- "1" — Steepest Descent (SD);
- "2" — FIRE (Fast Inertial Relaxation Engine) Algorithm[34].

Default: `1` (SD) — for VCNEB calculations; `2` (FIRE) — for structure relaxation

Format:

```
1 : optimizerType
```

▷ *variable* `optRelaxType`

Meaning: Structure relaxation mode:

- "1" — relax only atomic positions (with cell fixed), *e.g.* as in the classical NEB method;
- "2" — relax only cell lattice (used only for testing);
- "3" — full relaxation of atomic positions and cell.

Default: `3`

Format:

```
3 : optRelaxType
```

▷ *variable* `dt`

Meaning: Time step for structure relaxation.

Default: `0.05`

Format:

```
0.1 : dt
```

Note: If `dt` is very small, the calculations will be very slow. If `dt` is too large, the calculation will be unstable and may generate meaningless results.

▷ *variable* `ConvThreshold`

Meaning: Halting criteria condition for RMS (Root Mean Square forces) on images.

Default: `0.003` eV/Å

Format:

```
0.005 : ConvThreshold
```

▷ *variable* `VarPathLength`

Meaning: Criterion for path length between images for variable Image method. When the length between two neighbor images is larger than 1.5 times of `VarPathLength`, a new image will be added between the two images using linear interpolation; when less then 0.5 the value, the second image will be removed.

Default: The average path length between images of the initial pathway

Format:

```
0.3 : VarPathLength
```

▷ *variable* `K_min`

Meaning: Minimum spring constant, only used in variable-spring constant VCNEB (in eV/Å^2).

Default: `5`

Format:

```
3 : K_min
```

▷ *variable* `K_max`

Meaning: Maximum spring constant, only used in variable-spring constant VCNEB (in eV/Å^2).

Default: `5`

Format:

```
6 : K_max
```

▷ *variable* `Kconstant`

Meaning: Spring constant, Only used in fixed-spring constant VCNEB (in eV/Å^2).

Default: `5`

Format:

```
4 : Kconstant
```

▷ *variable* `optFreezing`

Meaning: Option for freezing the Image structure. Image structure will be frozen when `ConvThreshold` is achieved if enabled. Image structure freezing options:

- “0” — no Images freeze any time;
- “1” — freeze when `ConvThreshold` is achieved.

Default: `0`

Format:

```
1 : optFreezing
```

▷ *variable* `optMethodCIDI`

Meaning: Option for Climbing-Image (CI) and Descending-Image (DI) method. This method is only suggested to be used when you have a reasonable and well converged pathway. CI/DI-Image method options:

- "0" — CI/DI method not used;
- "1" — single CI method, only the highest energy or user-provided transition state (TS) will be used for CI;
- "-1" — single DI method, only the lowest energy or user-provided local minimum state (LM) will be used for DI;
- "2" — mixed multi-CI/DI method, the sequential numbers of TS and LM states need to be provided;

Default: `0`

Format:

```
1 :  optMethodCIDI
```

▷ *variable* `startCIDIStep`

Meaning: CI/DI method starting step number, only available when `optMethodCIDI`=1.

Default: `100`

Format:

```
200 :  startCIDIStep
```

▷ *variable* `pickupImages`

Meaning: Number of images to be picked up for CI/DI method.

Default: Image number of transition state and local minima state Images

Format:

```
% pickupImages
9 11 17
% EndPickupImages
```

Note: In this case, the 9^{th}, 11^{th} and 17^{th} images will be picked up for applying CI/DI-Image method. The image at transition state will evolve with CI method and the image at local minimum will evolve with DI method automatically.

▷ *variable* `FormatType`

Meaning: The format of structures in pathway output file, locates in `results1/PATH/`. Pathway structures output format:

- "1" — XCRYSDEN format (.xsf file);
- "2" — VASP POSCAR;
- "3" — XYZ format with lattice.

Default: `2`

Format:

```
1 : FormatType
```

▷ *variable* `PrintStep`

Meaning: Save the VNCEB restart files located in `results1/STEP/` every `PrintStep` steps.

Default: `1`

Format:

```
10 : PrintStep
```

Note: For forcefield codes, such as GULP, we suggest users to set `PrintStep`=10 to reduce time cost of saving the restart files.

Fig. 16 shows an example of use of the VCNEB method: phase transition mechanism and energy barrier with the $Ibam \rightarrow P6/mmm$ transition of BH at 168 GPa, here we obtained a $Pbcm$ intermediate phase.

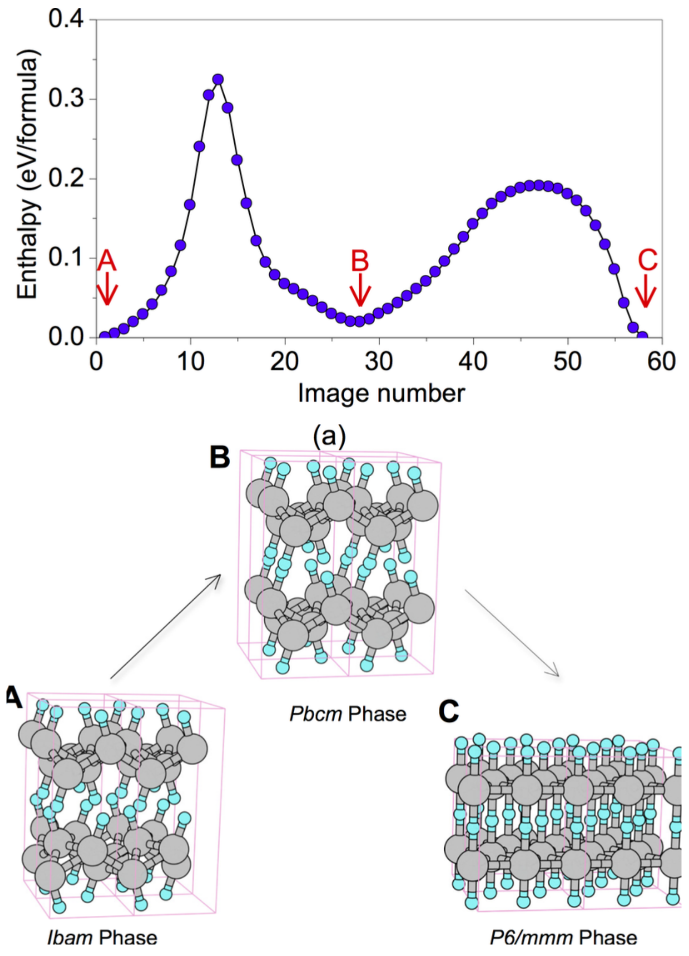


Figure 16: **The** $Ibam \to P6/mmm$ **transition of BH at 168 GPa[14]**. A $Pbcm$ intermediate phase is revealed. The saddle points on $Ibam \to Pbcm$ and $Pbcm \to P6/mmm$ segments have barriers of 0.32 and 0.19 eV/f.u., respectively.

6.2.2 How to set the initial pathway in the VCNEB calculation

The VCNEB method is very efficient for finding the phase transition path, but we must also carefully prepare the initial path. Cell rotations happen near the initial and final structures during the VCNEB calculation, where the pathway includes a lot of identical structures near the initial and finial images. We improved `Variable-Image-Number` method will prevent cell rotations automatically, which saves quite a lot of time.

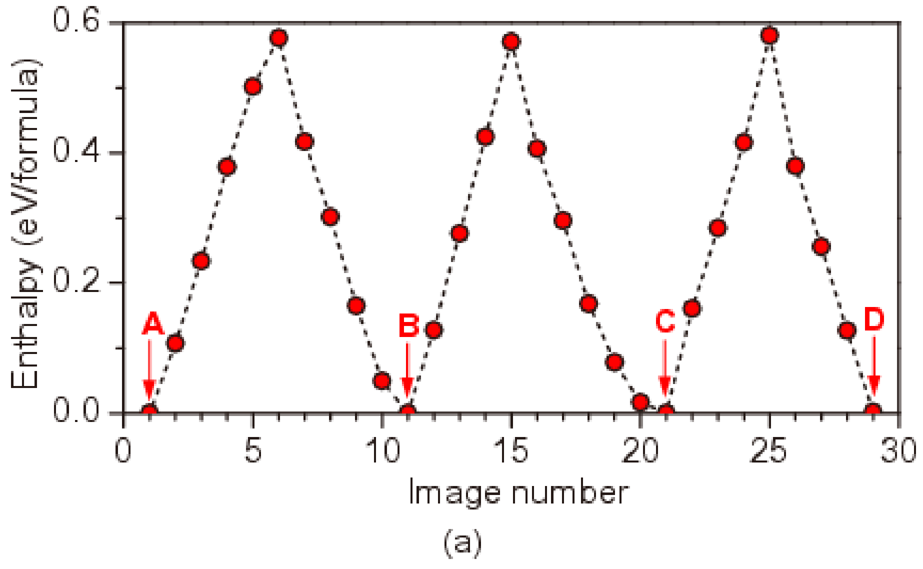
Alternatively, you can apply the rotation-avoiding technique before you apply the VCNEB method when generating the initial image set. The general 3×3 rotation matrix with Euler angles $R(\phi, \theta, \psi)$ and the lattice mirror operator $M(x, y, z)$ matrix are defined. Before performing a VCNEB calculation, the global numerical search in space of Euler angles and mirror operator are used to find the minimal lattice cell transformation distance Δh:

$$\Delta h = \left| h_{initial} - R(\phi, \theta, \psi) M(x, y, z) h_{final} \right| . \tag{10}$$

The rotation-avoiding lattice vector of the final image $\tilde{h}_{final}$ is assigned as the endpoint image:

$$\tilde{h}_{final} = R(\phi, \theta, \psi) M(x, y, z) h_{final} . \tag{11}$$

More important, we need to prevent the arbitrariness in assigning the atomic fractional coordinates $\mathbf{r}_v$ of the initial and finial images (correctly mapping the atoms at the initial and final structures). Otherwise, the calculation will be hard to converge or several identical paths can be found in a calculation, as shown in Fig. 17. For more complicated systems, you will get some unreasonable or messy pathways if you don't have a good initial pathway. Global numerical search for minimizing the distance between the atoms from two endpoint images helps the VCNEB method to reassign the atom sequence. The ability to automatically create model paths before the VCNEB calculation is crucial for the stability and convergence of the algorithm, and is a prerequisite for studying large and complex systems.



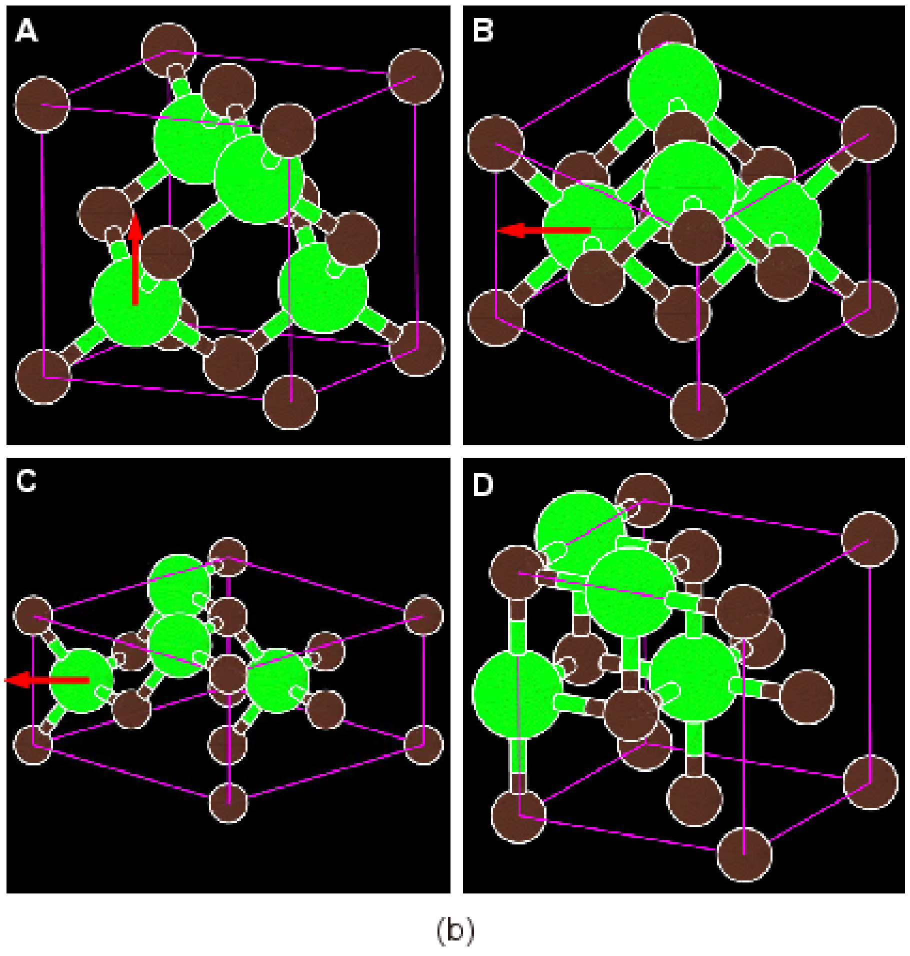


Figure 17: **Identical pathways found when setting up a "bad" initial image file**. The pathway is for the B3→B1 phase transition in GaN at the equilibrium pressure 45.0 GPa. At images 11 and 21, B1 and B3 structures in a monoclinic cell are found during the MEP optimization. The Ga atoms move along the arrow directions during the phase transition.

6.3 Transition path sampling (TPS) method

Transition path sampling (TPS) is a method for simulating rare events: a good example of a rare event is a transition of a system from one stable state to another; these occur due to rare fluctuations, which cannot be observed within typical timescales of molecular dynamics simulations. Examples include protein folding, chemical reactions and nucleation. TPS is a powerful method to study these phenomena. In short, TPS is a technique performing Monte-Carlo sampling over transition trajectories (each of which is a molecular dynamics trajectory).

6.3.1 Input options for TPS

The TPS method is only enabled with the LAMMPS and CP2K (will be available soon) codes at the moment.

To switch on the TPS mode, you have to:

1. Specify

   `TPS : calculationMethod`

2. Create a MD restart file `lammps.restart` for LAMMPS code or `cp2k.restart` for CP2K code in the `Seeds/` folder.

3. Create an order parameter calculation script or use USPEX's fingerprints as default. (See more details below).

4. Specify the following TPS options:

▷ *variable* `numIterations`

Meaning: Maximum number of TPS steps.

Default: `1000`

Format:

```
500 :  numIterations
```

▷ *variable* `speciesSymbol`

Meaning: Describes identities of all chemical species (atomic or molecular).

Default: No default

Format:

```
% speciesSymbol
CH4 Si O
% EndSpeciesSymbol
```

Note: In TPS, we use `speciesSymbol` to replace the `atomType` to support molecular species, together with `numSpecies`. For example, we can use

```
% numSpecies
1 648 1296
% EndNumSpecies

% speciesSymbol
CH4 Si O
% EndSpeciesSymbol
```

to study diffusion of a methane (CH_4) molecule in a silica zeolite (SiO_2). In this case, methane molecule will be considered as a whole particle in writing MD input files in LAMMPS.

▷ *variable* `mass`

Meaning: Masses for each type of species.

Default: Default value corresponding to the species

Format:

```
% mass
16.000
% EndMass
```

▷ *variable* `amplitudeShoot`

Meaning: Distribution amplitude of the momentum for A→B and B→A directions in shooting operation.

Default: `0.1 0.1`

Format:

```
% amplitudeShoot
0.12 0.12
% EndAmplitudeShoot
```

▷ *variable* `magnitudeShoot`

Meaning: Factor for increasing or decreasing the amplitude of magnitude distribution when succeeded or failed to have a new MD trajectory, respectively.

Default: `1.05 1.05`

Format:

```
% magnitudeShoot
1.0 1.05
% EndMagnitudeShoot
```

Note: Unless the user has system-specific experience, we strongly suggest to use this option to automatically adjust the distribution amplitude of the momentum.

▷ *variable* `shiftRatio`

Meaning: Ratio of applying a shifter operation after having a successful shooter operation.

Default: `0.1`

Format:

```
0.5 : shiftRatio
```

▷ *variable* `orderParaType`

Meaning: Method of order parameter calculation to distinguish different phases:

- "0" — user-defined method to calculate the order parameter, a script should be provided in `cmdOrderParameter`;
- "1" — fingerprint method (Oganov & Valle, 2009).

Default: No default

Format:

```
0 : orderParaType
```

▷ *variable* `opCriteria`

Meaning: Two values here specify the tolerable degree of similarity to the starting and ending states, respectively. If one or both ends of the trajectory lead to different end structures, such a trajectory will be rejected in the TPS method.

Default: No default

Format:

```
% opCriteria
0.995 0.995
% EndOpCriteria
```

Note: If you use the fingerprint method, the larger system you have, the more strict criteria you should have. For example, we should set `opCriteria`=[0.995, 0.995] with 1,000 atoms; and `opCriteria`=[0.998, 0.998] in system more than 30,000 atoms when studying the fcc→hcp phase transition.

▷ *variable* `cmdOrderParameter`

Meaning: User-defined command to calculate the order parameter. It is not needed if you use the fingerprint method.

Default: No default

Format:

```
% cmdOrderParameter
./extractOp.sh
% EndCmdOrderParameter
```

▷ *variable* `cmdEnthalpyTemperature`

Meaning: User-defined command to extract the enthalpy and temperature from MD results.

Default: No default

Format:

```
% cmdEnthalpyTemperature
./extractHT.sh
% EndCmdEnthalpyTemperature
```

▷ *variable* `orderParameterFile`

Meaning: File name to store the order parameter history in a single MD calculation for TPS to read.

Default: `fp.dat`

Format:

```
op.dat :  orderParameterFile
```

▷ *variable* `enthalpyTemperatureFile`

Meaning: File name to store the enthalpy & temperature history in a single MD calculation for TPS to read or user-defined script to analyze.

Default: `HT.dat`

Format:

```
HT0.dat :  enthalpyTemperatureFile
```

▷ *variable* `trajectoryFile`

Meaning: File name to store the MD trajectory. This name should be consistent with the MD trajectory output file name from the calculation.

Default: `traj.dat`

Format:

```
trajectory.xyz :  trajectoryFile
```

▷ *variable* `MDrestartFile`

Meaning: File name to store the MD restart file from the calculation for TPS to read. This name should be consistent with the MD restart file name from the calculation.

Default: `traj.restart`

Format:

```
lammps0.restart :  MDrestartFile
```

7 Online utilities

We have created a number of useful online utilities, which can be used for preparing USPEX input and for post-processing. The utilities are available at:

`http://uspex-team.org/online_utilities/`

Below you can find information about each one of them.

7.1 Structure characterization

Here we have 4 utilities:

- Fingerprints — the utility calculates and plots fingerprint function, which is a crystal structure descriptor, a 1D-function related to the pair correlation function and diffraction patterns. It does not depend on absolute atomic coordinates, but only on interatomic distances. Small deviations in atomic positions will influence the fingerprint only slightly, *i.e.* it is numerically robust.
- Multifingerprint — the utility calculates average quasi-entropy, A-order(average atomic order parameter) and S-order(whole-structure order parameter) for a set of structures. Also it filters unique structures by cosine distances difference ≥ 0.003, identifies the symmetry of these structures and lists them in the `uniq_gatheredPOSCARS` file.
- POSCAR2CIF — determines space group and prepares a `CIF` file from a `POSCAR` file.
- CIF2POSCAR — prepares a `POSCAR` file from a `CIF` file.
- XSF2POSCAR — prepares a `POSCAR` file from a `XSF` (XCRYSDEN) file.

7.2 Properties calculations

Here we have 2 utilities:

- Hardness — the utility is to calculate hardness based on the Lyakhov-Oganov model.
- EELS — the utility calculates the Electron Energy Loss Spectrum (EELS). Written by Priya Johari.

7.3 Molecular crystals

Here we have 2 utilities:

- MOL precheck — the utility allows you to check `MOL_1` files before running USPEX for molecular crystals (`calculationType`=310/311/110).
- Zmatrix — the utility converts `XYZ` file to USPEX `MOL_1` file.

7.4 Surfaces

Substrate — a program which prepares a substrate from a `POSCAR/CIF` file, given Miller indices, thickness of the layer, and shift. The resulting `POSCAR` file can be used for `calculationType`=200/201 as a substrate for surface calculations.

7.5 Miscellaneous

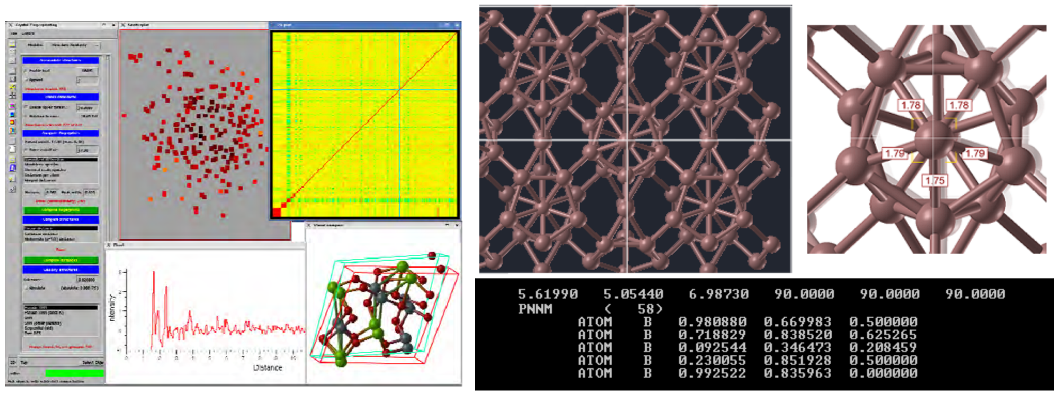
Here we have the following:

- Input generator — USPEX `INPUT.txt` generator. The utility can help beginners to create a correct input for USPEX calculations.
- Volume estimation — the utility estimates volumes of non-molecular and molecular crystals for USPEX (for `INPUT.txt` file).
- USPEX manual — online version of this manual.
- USPEX examples — archives with USPEX examples.

8 Frequently asked questions

8.1 How can I visualize the results?

USPEX produces a large set of numbers (structures, energies, *etc*). Post-processing, or analysis of the data, is extremely important. Analysis of these data "by hand" can be quite tedious and time-consuming. USPEX benefits from an interface specifically developed for USPEX by Mario Valle to read and visualize USPEX output files using his STM4 visualization toolkit[35], which includes analysis of thousands of structures in a matter of a few minutes, determination of structure-property correlations, analysis of algorithm performance, quantification of the energy landscapes, state-of-the-art visualization of structures, determination of space groups, *etc.*, including preparation of movies showing the progress of the simulation! Fig. 18 shows typical figures produced by STM4. To use STM4, you need to have AVS/Express installed on your computer. AVS/Express is not public domain and requires a license. STM4 is available at http://mariovalle.name/STM4.



```
    5.61990   5.05440   6.98730   90.0000   90.0000   90.0000
    PNNM        <    58>
             ATOM    B    0.980880  0.669983  0.500000
             ATOM    B    0.718829  0.838520  0.625265
             ATOM    B    0.092544  0.346473  0.208459
             ATOM    B    0.230055  0.851928  0.500000
             ATOM    B    0.992522  0.835963  0.000000
```

Figure 18: **STM4 interface for USPEX.**

Alternatively, you can visualize USPEX results with other software, *e.g.*, VESTA, which can USPEX structure files directly.

8.2 How can I avoid trapping?

First, use a sufficiently large population size. Second, USPEX by default uses a powerful fingerprint niching method. Anything that increases diversity of the population will reduce the chances of trapping in a local minimum. To make sure that your simulation is not trapped, it is useful to run a second simulation with different parameters. Another powerful trick to avoid trapping is the antiseed technique.

8.3 What is a single-block calculation?

The single-block feature was introduced in USPEX 9.3.9, and enables USPEX users to run structure predictions with a variable number of formula units of the same composition. For example:

```
% atomType
Si O
% EndAtomType

% numSpecies
1 2
% EndNumSpecies

12 : minAt
24 : maxAt
```

This means we sample structures of compound SiO_2 (with the atomic ratio of 1:2) with a variable number of formula units with 12–24 atoms.

Starting from USPEX 9.4.1, the single block feature has been moved from `calculationType` = 301/311 to 300/310. The settings are still the same, users just need to set up the `minAt`, `maxAt` and `numSpecies` keywords.

Currently, one can use this feature in 300, 310 and -200 (since 2015 October).

8.4 How to predict structures based on known fragments?

Sometimes you already know that your structure is based on packing of some well defined motifs. You can tell the program to generate only structures based on the packing 'fake molecules'. Meanwhile, you still use the standard approach to generate child structures (such as heredity and mutation) which treat the structure as atomic crystals, and thus can break the predefined motif. To activate the function, you just prepare an ordinary INPUT.txt,

```
% atomType
B
% EndAtomType

% numSpecices
48
% EndNumSpecices
```

Suppose you want to generate structures base on boron icosahedra, an additional MOL_1 file is needed

```
B12_[4]
Number of atoms: 12
B  -1.591325  -0.618615  -0.217220     0 0 0   0
B  -0.574110  -0.095870  -1.619670     1 0 0   0
B  -1.211325   1.134555  -0.455665     2 1 0   0
B  -1.158010   0.414740   1.203810     3 2 1   0
B  -0.487865  -1.260560   1.065420     4 3 1   0
B  -0.126980  -1.576135  -0.679575     5 4 1   0
B   0.487845   1.260560  -1.065430     3 2 1   0
B   0.127000   1.576130   0.679585     4 3 1   0
B   0.574120   0.095870   1.619670     5 4 1   0
B   1.211315  -1.134555   0.455685     6 5 1   0
B   1.158015  -0.414735  -1.203800     2 6 1   0
B   1.591320   0.618615   0.217195     7 8 3   0
```

Its format has been described in Section 5.3. The only difference is that we need to put some additional information in the header. [4] here tells that 4 MOL_1 (B12) will be used, which is consistent with 48 B atoms described in INPUT.txt. The consistency is required. Otherwise, the fragment feature won't be activated, even though you put addition MOL_x files in the INPUT.txt. If you set everything right, you will find the messages that this feature is used from OUTPUT.txt.

For 2D crystals, you need to specify two thicknesses in thic case.

```
4.0   : thicknessS (it specifies the overall thickness of 2D crystal )
0.0   : thicknessB (it specifies the thickness measured by the molecular centers)
```

Currently, one can use this feature in 300 and -200. Note that you cannot use single block and fragement feature simutaneously. It is highly recommended to use it with the fixed cell mode, when the cell parameters are available.

8.5 How do I use the seed technique?

This technique is useful, if instead of starting with random structures, you would like to input some structures that you already know for the compound or related materials. Just create a file `Seeds/POSCARS` for the next generation, or `Seeds/POSCARS_gen` (`gen` is the generation number) for the specific generation of an USPEX calculation, in the format of concatenated `POSCAR` files in VASP5 format. Don't miss letter "`S`" in the file name.

Example:

```
EA33 2.69006 5.50602 4.82874 55.2408 73.8275 60.7535 no SG
1.0
2.690100  0.000000  0.000000
```

```
2.690100  4.804100  0.000000
1.344900  2.402100  3.967100
Mg Al O
1  2  4
Direct
0.799190  0.567840  0.859590
0.793520  0.230950  0.544750
0.793540  0.916090  0.174450
0.050972  0.816060  0.859610
0.172230  0.194810  0.859600
0.438250  0.655170  0.406880
0.438230  0.202440  0.312330
EA34 7.61073 2.85726 2.85725 60.0001 79.1809 79.1805 no SG
1.0
7.610700  0.000000  0.000000
0.536350  2.806500  0.000000
0.536330  1.352000  2.459300
Mg Al O
1  2  4
Direct
0.708910  0.507440  0.068339
0.374050  0.285730  0.846630
0.023663  0.069185  0.630090
0.889560  0.780560  0.341460
0.350470  0.626920  0.187820
0.597290  0.211310  0.772210
0.116440  0.371590  0.932500
```

One can add seeds at any time during the calculation. USPEX will look for new seeds at the beginning of each generation. The corresponding information will be recorded to `results1/Seeds_history` and the seeds files (`POSCARS` or `POSCARS_gen`) will be kept as `POSCARS_gen` in `Seeds/` folder.

Whenever seeds are added, we advise users to check the `results1/Seeds_history` and `Warnings` files. There will be a warning message "`Meet a problem when reading Seeds - ...`" if your seeds are problematic. When an error appears in the seeds file, such as missing lines, the structures after the error point will not be added.

Note: Make sure you specified all atomic symbols at the 6th line of each structure. For example, to add the $P6_3/mc$ H_2 structure to a H-O variable-composition calculation, you should edit the file as:

```
H_I-P63/mc
1
4.754726  -2.74514  0.000000
-0.00000  5.490285  0.000000
0.000000  0.000000  4.508715
H
16
Direct
```

```
...
(the atomic positions information is omitted here)
```

8.6 How do I play with the compositions?

For variable-composition and single block calculations, as soon as the calculation starts, it produces a file `Seeds/compositions` with all possible compositions, from which the code randomly takes compositions for the random structure generator. You can edit this file, leaving the compositions you are most interested in — only these compositions will be used for random structure production in the second and subsequent generations. `Seeds/compositions` file lists the numbers of atoms of each type in the cell, *e.g.*, for the C-O system:

```
8 0
0 8
2 4
```

means that you are interested in randomly producing C_8, O_8, and C_2O_4 structures. Other compositions will be sampled too, thanks to the heredity and transmutation operators.

When you want to generate structures with specific compositions, you can use the anti-compositions feature — write the list of all unwanted compositions to the file named `Seeds/Anti-compositions`. There are three ways to do so:

1. For all unwanted compositions with the same ratios, you can write stoichiometric ratio to ban these compositions. For example, you can use “`1 2 1`” to ban all the composition with the same ratio, such as “`1 2 1`”, “`2 4 2`”, “`3 6 3`” and so on.
2. Only for the specific composition, but not for other compositions with the same ratio. You can write the compositions with a minus sign. For example, you can use “`-3 2 0`” or “`3 -2 0`” to ban the “`3 2 0`” composition, but not to ban “`6 4 0`” or “`9 6 0`” composition. (Notice: “`3 2 -0`” does not work for this case).
3. For all single/binary/ternary compounds. If you don’t want to sample all single/binary/ternary compounds, just write the keyword `single/binary/ternary` in `Anti-compositions` file.

Example:

```
 single
 binary
 1 1 2
-2 2 1
```

If you don't clearly know what you are doing, please leave `Anti-compositions` file empty. For more information about the compositions you don't want, you can have a look at `results1/compositionStatistic` file.

Note:

- Even if `compositions` or `Anti-compositions` files exist before the calculation starts, they will be ignored. `Anti-compositions` file will be renamed to a backup file `Anti-compositions-back`. Therefore, please edit `compositions` or `Anti-compositions` files after the calculation starts.
- Please also be aware, that in USPEX calculations with compositional blocks, the compositions usually mean the numbers of these blocks. Therefore, to have the correct format of `Anti-compositions` file, please check `compositions` file first.

8.7 How do I set up a passwordless connection from a local machine to a remote cluster?

There are two ways to solve this problem:

1. SSH login without password.

Note: this part based on follows article: `http://linuxproblem.org/art_9.html`

Your aim: You want to use OpenSSH to enable automatic job submission. Therefore you need an automatic login from `hostA` / `userA` to `hostB` / `userB`. You don't want to enter any passwords, because you want to call ssh from within a shell script.

How to do it: First log in on A as user A and generate a pair of authentication keys. Do not enter a passphrase:

```
userA@hostA:~> ssh-keygen -t rsa  Generating public/private rsa key
pair.
Enter file in which to save the key (/home/userA/.ssh/id_rsa):
Created directory '/home/userA/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/userA/.ssh/id_rsa.
Your public key has been saved in /home/userA/.ssh/id_rsa.pub.
The key fingerprint is:
3e:4f:05:79:3a:9f:96:7c:3b:ad:e9:58:37:bc:37:e4 userA@hostA
```

Now use ssh to create a directory ∼/.ssh as `userB` on `hostB` with `portB`. (The directory may already exist, which is fine):

```
userA@hostA:~> ssh -p portB userB@hostB 'mkdir -p .ssh' userB@hostB's
password:
```

Finally append A's new public key to `userB@hostB`: .ssh/authorized_keys and enter B's password one last time:

```
userA@hostA:~> cat .ssh/id_rsa.pub | ssh -p portB userB@hostB'cat >>
.ssh/authorized_keys'.
 userB@hostB's password:
```

From now on you can log into `hostB` as `userB` from `hostA` as `userA` without password:

```
userA@hostA:~> ssh -p portB userB@hostB
```

2. You will need to copy the public key from your local machine (directory `./ssh` or `./ssh2`) to the remote cluster. Here is the list of commands you need to execute:

```
local # ssh-keygen -t dsa
local # scp ~/.ssh2/id_dsa.pub oganov@palu.cscs.ch:~/.ssh/tmp.pub
remote # cd ~/.ssh/
remote # ssh-keygen -f tmp.pub -i >> authorized_keys
remote # rm tmp.pub
```

8.8 How do I set up a calculation using a job submission script?

To set up a job submission script, we expect users to know some basic knowledge of python programing and your job submission systems.

There are two modes for job submission: **local** submission or **remote** submission, depending on whether you submit *ab initio* calculations to the local machine where you run USPEX, or to a remote supercomputer.

8.8.1 Step 1: Configuring files in `Submission/` folder

Case I: Local submission.

Please edit in `INPUT.txt` file the following tag:

```
1   : whichCluster (0: no-job-script, 1: local submission, 2: remote submission)
```

Then, it is necessary to run ssh server on your local machine. USPEX will connect to it and run ab-initio code via ssh.

Then, go to the directory `Submission/`, where you need to edit two files: `submitJob_local.py` and `checkStatus_local.py`.

One can find the detailed instructions in these files. In general, one just needs to tell USPEX how to submit the job and check if the job has completed or not.

In `submitJob_local.py`:

```
from subprocess import check_output
import re
import sys


def submitJob_local(index : int, commnadExecutable : str) -> int:
    """
    This routine is to submit job locally
    One needs to do a little edit based on your own case.

    Step 1: to prepare the job script which is required by your supercomputer
    Step 2: to submit the job with the command like qsub, bsub, llsubmit, .etc.
    Step 3: to get the jobID from the screen message
    :return: job ID
    """

    # Step 1
    myrun_content = ''
    myrun_content += '#!/bin/sh\n'
    myrun_content += '#SBATCH -o out\n'
    myrun_content += '#SBATCH -p cpu\n'
    myrun_content += '#SBATCH -J USPEX-' + str(index) + '\n'
    myrun_content += '#SBATCH -t 06:00:00\n'
    myrun_content += '#SBATCH -N 1\n'
    myrun_content += '#SBATCH -n 8\n'
    # myrun_content += 'cd ${PBS_O_WORKDIR}\n' check this, must have /cephfs suffix with
    SBATCH in my case
    myrun_content += 'mpirun vasp_std > log\n'
    with open('myrun', 'w') as fp:
        fp.write(myrun_content)

    # Step 2
    # It will output some message on the screen like '2350873.nano.cfn.bnl.local'
    output = str(check_output('sbatch myrun', shell=True))

    # Step 3
    # Here we parse job ID from the output of previous command
    jobNumber = int(re.findall(r'\d+', output)[0])
    return jobNumber


if __name__ == '__main__':
    import argparse
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', dest='index', type=int)
    parser.add_argument('-c', dest='commnadExecutable', type=str)
    args = parser.parse_args()

    jobNumber = submitJob_local(index=args.index, commnadExecutable=args.
    commnadExecutable)
    print('CALLBACK ' + str(jobNumber))
```

In `checkStatus_local.py`:

```
import argparse
import glob
import os

from subprocess import check_output

_author_ = 'etikhonov'

```

```
def checkStatus_local(jobID : int) -> bool:
    """
    This function is to check if the submitted job is done or not
    One needs to do a little edit based on your own case.
    1   : whichCluster (0: no-job-script, 1: local submission, 2: remote submission)
    Step1: the command to check job by ID.
    Step2: to find the keywords from screen message to determine if the job is done
    Below is just a sample:
    -------------------------------------------------------------------------------
    Job id                    Name              User              Time Use S Queue
    ------------------------- ----------------- ----------------- -------- - -----
    2455453.nano              USPEX             qzhu              02:28:42 R cfn_gen04
    -------------------------------------------------------------------------------
    If the job is still running, it will show as above.

    If there is no key words like 'R/Q Cfn_gen04', it indicates the job is done.
    :param jobID:
    :return: doneOr
    """

    # Step 1
    output = str(check_output('qstat {}'.format(jobID), shell=True))
    # Step 2
    doneOr = True
    if ' R ' in output or ' Q ' in output:
        doneOr = False
    if doneOr:
        for file in glob.glob('USPEX*'):
            os.remove(file)  # to remove the log file
    return doneOr

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-j', dest='jobID', type=int)
    args = parser.parse_args()

    isDone = checkStatus_local(jobID=args.jobID)
    print('CALLBACK ' + str(int(isDone)))
```

Case II: Remote submission.

Please edit in `INPUT.txt` file the following tag:

```
2       : whichCluster (default 0, 1: local submission; 2: remote submission)
```

Finally, go to the directory `Submission/`, where you need to edit two files: `submitJob_remote.py` and `checkStatus_remote.py`

In `submitJob_remote.py`:

```
import argparse
import os
import re

from subprocess import check_output


def submitJob_remote(workingDir : str, index : int, commandExecutable : str) -> int:
    """
    This routine is to submit job to remote cluster
    One needs to do a little edit based on your own case.
    Step 1: to prepare the job script which is required by your supercomputer
    Step 2: to submit the job with the command like qsub, bsub, llsubmit, .etc.
    Step 3: to get the jobID from the screen message

    :param workingDir: working directory on remote machine
    :param index: index of the structure.
    :param commandExecutable: command executable for current step of optimization
    :return:
    """

    # Step 1
    # Specify the PATH to put your calculation folder
    Home = '/home/etikhonov' # 'pwd' of your home directory of your remote machine
    Address = 'rurik'  # your target server: ssh alias or username@address
    Path = Home + '/' + workingDir + '/CalcFold' + str(index) # Just keep it
    run_content = ''
    run_content += '#!/bin/sh\n'
    run_content += '#SBATCH -o out\n'
    run_content += '#SBATCH -p cpu\n'
    run_content += '#SBATCH -J USPEX-' + str(index) + '\n'
    run_content += '#SBATCH -t 06:00:00\n'
    run_content += '#SBATCH -N 1\n'
    run_content += '#SBATCH -n 8\n'
    run_content += 'cd /cephfs'+ Path + '\n'
    run_content += commandExecutable + '\n'

    with open('myrun', 'w') as fp:
        fp.write(run_content)

    # Create the remote directory
    # Please change the ssh/scp command if necessary.
    try:
        os.system('ssh -i ~/.ssh/id_rsa ' + Address + ' mkdir -p ' + Path)
    except:
        pass

    # Copy calculation files
    # add private key -i ~/.ssh/id_rsa if necessary
    os.system('scp POSCAR   ' + Address + ':' + Path)
    os.system('scp INCAR    ' + Address + ':' + Path)
    os.system('scp POTCAR   ' + Address + ':' + Path)
    os.system('scp KPOINTS  ' + Address + ':' + Path)
    os.system('scp myrun ' + Address + ':' + Path)

    # Step 2
    # Run command
    output = str(check_output('ssh -i ~/.ssh/id_rsa ' + Address + ' qsub ' + Path + '/
    myrun', shell=True))

    # Step 3
    # Here we parse job ID from the output of previous command
    jobNumber = int(re.findall(r'\d+', output)[0])
    return jobNumber


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', dest='index', type=int)
    parser.add_argument('-c', dest='commnadExecutable', type=str)
    parser.add_argument('-f', dest='workingDir', type=str)
    args = parser.parse_args()

    jobNumber = submitJob_remote(workingDir=args.workingDir, index=args.index,
    commnadExecutable=args.commnadExecutable)
    print('CALLBACK ' + str(jobNumber))
```

In `checkStatus_remote.py`:

```
import argparse
import os

from subprocess import check_output

def checkStatus_remote(jobID : int, workingDir : str, index : int) -> bool:
    """
    This routine is to check if the submitted job is done or not
    One needs to do a little edit based on your own case.
    Step1: Specify the PATH to put your calculation folder
    Step2: Check JobID, the exact command to check job by jobID
    :param jobID:
    :param index:
    :param workingDir:
    :return:
    """
    # Step 1
    Home = '/home/etikhonov'  # 'pwd' of your home directory of your remote machine
    Address = 'rurik'  # Your target supercomputer: username@address or ssh alias
    # example of address: user@somedomain.edu -p 2222
    Path = Home + '/' + workingDir + '/CalcFold' + str(index)  # just keep it

    # Step 2
    output = str(check_output('ssh ' + Address + ' qstat ' + str(jobID), shell=True))
    # If you using full adress without ssh alias, you must provide valid ssh private key
    like there:
    # output = str(check_output('ssh -i ~/.ssh/id_rsa ' + Address + ' /usr/bin/qstat ' +
    str(jobID), shell=True))

    if not ' R ' in output or not ' Q ' in output:
        doneOr = True
        # [nothing, nothing] = unix(['scp -i ~/.ssh/id_rsa ' Address ':' Path '/OUTCAR ./
    ']) %OUTCAR is not necessary by default
        os.system('scp ' + Address + ':' + Path + '/OSZICAR ./')  # For reading enthalpy/
    energy
        os.system('scp ' + Address + ':' + Path + '/CONTCAR ./')  # For reading
    structural info
        # Edit ssh command as above!
    else:
        doneOr = False
    return doneOr


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('-j', dest='jobID', type=int)
    parser.add_argument('-i', dest='index', type=int)
    parser.add_argument('-f', dest='workingDir', type=str)
    args = parser.parse_args()

    isDone = checkStatus_remote(jobID=args.jobID, workingDir=args.workingDir, index=args.
    index)
    print('CALLBACK ' + str(int(isDone)))
```

9 Appendices

9.1 List of Examples

- `EX01-3D_Si_vasp`: Silicon (8 atoms/cell) at zero pressure. Variable-cell DFT calculation using VASP, PBE96 functional. Many thanks to G. Kresse for permission to include his PAW files (`POTCAR`) in our distribution.
- `EX02-3D_MgAl2O4_gulp`: $MgAl_2O_4$ (28 atoms/cell) at 100 GPa pressure. Variable-cell calculation using Buckingham potentials, GULP code. Beware that for reliable results, you should better do *ab initio* calculations.
- `EX03-3D-const_cell_MgSiO3_gulp`: this example shows how to do structure prediction when you know cell parameters. $MgSiO_3$ (20 atoms/cell) with Buckingham potentials, GULP code. Cell parameters correspond to post-perovskite. The discovery of post-perovskite (*Oganov & Ono, Nature 2004; Murakami et al., Science 2004*) was a major breakthrough in Earth sciences.
- `EX04-3D_C_lammps`: this example shows how to do crystal structure prediction using USPEX together with the LAMMPS code. In this a simple example: 8 carbon atoms, and Tersoff potential.
- `EX05-3D_Si_atk`: Example of crystal structure prediction of Si with 8 atoms/cell using the density-functional tight binding approximation and ATK code.
- `EX06-3D_C_castep`: DFT-based prediction of the crystal structure of carbon with 8 atoms/cell at 10 GPa, using the CASTEP code.
- `EX07-2D_Si_vasp`: prediction of the 2D-crystal of silicon using DFT and VASP. Simple and powerful.
- `EX08-0D_LJ_gulp`: Nanoparticle structure prediction. Lennard-Jones nanoparticle with 30 atoms, using the GULP code.
- `EX09-3D-molecules_CH4_vasp`: methane with 4 molecules/cell, at the pressure of 20 GPa. DFT, VASP. Molecule is described in the file `MOL_1`.
- `EX10-3D-molecules_CH4_dmacrys`: methane with 8 molecules/cell, with forcefield and DMACRYS code, at normal pressure. Molecule is described in the file `MOL_1`, but note its slightly unusual format for DMACRYS calculations. Please put executables `dmacrys`, `neighcrys-pp`, `neighcrys-vv` in the `Specific/` folder.
- `EX11-3D-molecules_urea_tinker`: urea with 2 molecules/cell, with forcefield and TINKER code, at normal pressure. Molecule is described in the file `MOL_1`.
- `EX12-3D_varcomp_LJ_gulp`: Lennard-Jones binary system with fake "Mo" and "B" atoms, GULP, and variable-composition USPEX (*Lyakhov and Oganov, 2010*).

- `EX13-3D_special_quasirandom_structure_TiCoO`: USPEX can easily find the most disordered (or the most ordered) alloy structure. Here, this is shown for $Ti_xCo_{(1-x)}O$. You need to specify the initial structure in `Seeds/POSCARS` and use only the permutation operator. In this case, you don't need to use any external codes. In this example, we optimize (minimize) the structural order (*Oganov and Valle (2009); Lyakhov, Oganov, Valle (2010)*) without relaxation (`abinitioCode` = 0). Seed structure (supercell of Ti-Co-O-structure) is permutated to find the structure the minimum/maximum order. Minimizing order in this situation, one gets a generalized version of the "special quasirandom structure".

- `EX14-GeneralizedMetadynamics_Si_vasp`: simple example of a powerful capability to find complex low-energy structures starting with a simple seed structure (*Zhu et al, 2013*). Silicon, up to 16 atoms/cell, DFT, VASP. Pay special attention to `INCAR` files. Best of all, just keep the files that you see here, changing only `ENCUT`, perhaps `SIGMA`. Evolutionary metadynamics not only predicts low-energy structures, but also gives an idea of transition mechanisms between crystal structures.

- `EX15-VCNEB_Ar_gulp`: example of a variable-cell nudged elastic band (*VCNEB: Qian et al., 2013*) calculation of the fcc-hcp transition in a model system, argon, at 0 GPa pressure. Lennard-Jones potential, GULP code.

- `EX16-USPEX-performance_SrTiO3_gulp`: $SrTiO_3$ (50 atoms/cell) at zero pressure. Variable-cell calculation using Buckingham potentials, GULP code. Running this example you can see that even for such a relatively large system USPEX code scores a >90% success rate and remarkable efficiency. This contrasts with a 7-12% success rate reported for the same system and using the same potential by Zurek & Lonie. Clearly, USPEX outperforms the poor reimplementation of our method by Zurek and Lonie. We have witnessed excellent performance of our code also for much larger systems.

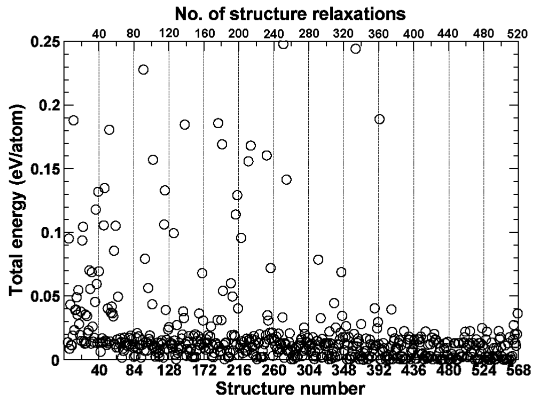
- `EX17-3D_DebyeTemp_C_vasp`: example of optimization of the elasticity-related properties (bulk or shear moduli, Poisson ratio, Chen-Niu hardness, or Debye temperature). In this example, we maximize the Debye temperature of carbon using the VASP code.

- `EX18-3D_varcomp_ZnOH_gulp`: as you know, USPEX has unique capabilities for variable-composition searches. This example shows a pretty challenging case — variable-composition calculation for the ternary system Zn-O-H. This calculation uses a ReaxFF forcefield in GULP code. USPEX can do calculations for any number of components — *e.g.* quaternary, quinternary, *etc.* systems are within its reach. Of course, the more components you have, the more expensive (and the more risky) your calculation is. No reference results at the moment.

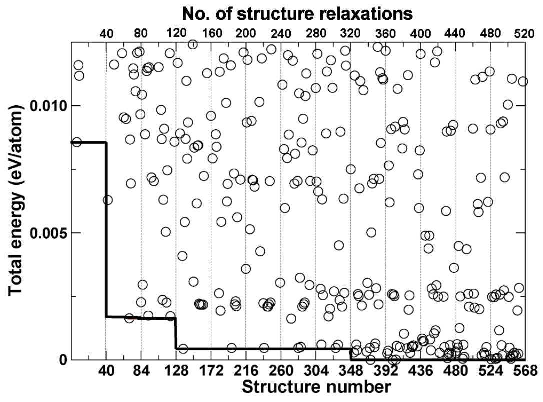
- `EX19-Surface-boron111`: Prediction of (111) surface reconstruction of alpha-boron, with variable number of atoms (Zhou *et al.*, Phys. Rev. Lett. 113, 176101 (2014)).

- `EX20-0D_Cluster_C60_MOPAC`: Cluster structure prediction (000) for C_{60} using MOPAC.
- `EX21-META_MgO_gulp`: Evolutionary metadynamics, with GULP code and Buckingham potentials, MgO with 8 atoms/cell. Starting structure is of rocksalt type, and evolutionary metadynamics finds a number of low-energy structures and structural relations.
- `EX22-GEM_MgO_gulp`: Generalized evolutionary metadynamics, with GULP code and Buckingham potentials. Starting structure is of rocksalt type, with 8 atoms/cell, the calculation is allowed to increase system size up to 16 atoms/cell, and generalized evolutionary metadynamics (GEM) finds a number of low-energy structures and structural relations.
- `EX23-MgO_surface`: Prediction of surface reconstruction of magnesium-oxygen, with fixed number of atoms.
- `EX24-SingleBlock_Magnetic_Fe3C_VASP`: Magnetic structure prediction (300) for Fe_3C in single block using VASP.
- `EX25-3D-C8-DFTB`: Structure prediction (300) for C (8 atoms/cell) using DFTB+ with 3ob-3-1 set. The energy differnet between graphite and diamond seems to be overestimated by the current DFTB parameter set.
- `EX26-Ar_TPS`: TPS calculation for the phase tranformation in Ar hcp and fcc solid of 8000 atoms at 40K under 1 atmosphere.
- `EX27-3D-P2_FHIaims`: Structure prediction (300) for P (2 atoms/cell) using FHI-aims.
- `EX28-0D-Cluster_Cu9_FHIaims`: Cluster structure prediction (000) for Cu_9 using FHI-aims.
- `EX29-Si_gap-maximize_singleblock`: Structure prediction (300) of silicon with optimization of the band gap using a meta-GGA functional.
- `EX30_Pd_111-oxidation`: Reconstructions involving foreign species on elemental surfaces (such as PdO@Pd(111) surface).
- `EX31_2D_varcomp_SnS_VASP`: Variable composition search for stable 2D-crystal (layer) Sn-S phases. Example is provided by Z.A.
- `EX32_pmpaths`: Generating by the pmpaths 5 different pathways of diamond-graphite transition. Example is provided by V. Stevanovic.

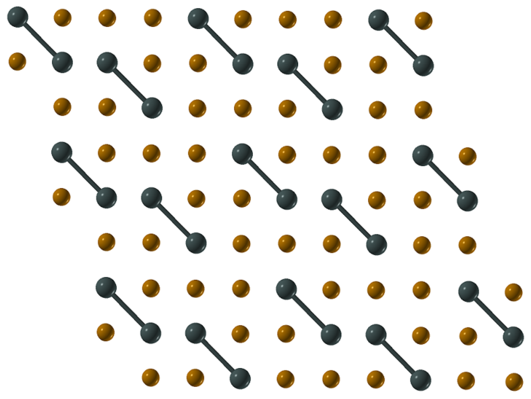
9.2 Test runs



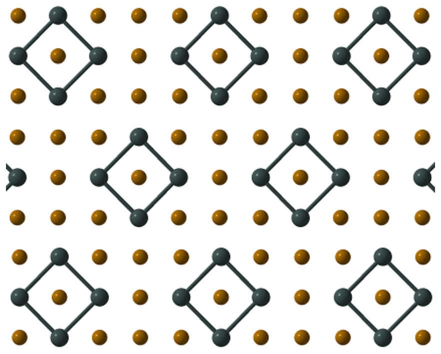
(a)



(b)



(c)



(d)

Figure 19: **Evolutionary structure search for Au_8Pd_4.** a, b — evolution of the total energy (for clarity, panel (b) zooms in on the lowest-energy region of the same data set), c — the lowest-energy structure found in our evolutionary simulation, and d — the lowest-energy structure found by cluster expansion calculations of Zunger. Note that our structure (c) is the lowest-energy known structure for this compound. This establishes the power of our method (even in its ancient, 2007, version).

9.3 Sample INPUT.txt files

9.3.1 Fixed-composition USPEX calculation (calculationType=300):

```
PARAMETERS EVOLUTIONARY ALGORITHM
% Example of the short input, using most options as defaults

% atomType
Mg Al O
% EndAtomType

% numSpecies
2 4 8
% EndNumSpecies

 50    : numGenerations
 50.0  : ExternalPressure

% abinitioCode
3 3 3 3 3
% ENDabinit

% commandExecutable
gulp < input > output
% EndExecutable
```

9.3.2 Variable-composition USPEX calculation (calculationType=301):

```
USPEX : calculationMethod (USPEX, VCNEB, META)
301   : calculationType (dimension: 0-3; molecule: 0/1; varcomp: 0/1)
1     : AutoFrac

% atomType
Mo B
% EndAtomType

% numSpecies
1 0
0 1
% EndNumSpecies

80    : populationSize
200   : initialPopSize
60    : numGenerations
20    : stopCrit

11    : firstGeneMax
8     : minAt
18    : maxAt

% abinitioCode
3 3 3
% ENDabinit

% commandExecutable
gulp < input > output
% EndExecutable
```

9.3.3 Evolutionary metadynamics (`calculationMethod`=META):

```
META   : calculationMethod (USPEX, VCNEB, META)
301    : calculationType (dimension: 0-3; molecule: 0/1; varcomp: 0/1)

0.0001 : ExternalPressure

16     : maxAt
2.0    : minVectorLength
8.0    : maxVectorLength

15     : populationSize
40     : numGenerations
3.0    : mutationDegree
250.0  : GaussianHeight
0.3    : GaussianWidth
2      : FullRelax

abinitioCode
1 1 1 (1 1)
ENDabinit

% KresolStart
0.12 0.10 0.09 0.10 0.08
% Kresolend

% commandExecutable
mpirun -np 4 vasp > log
% EndExecutable
```

9.3.4 VCNEB calculation (calculationMethod=VCNEB):

```
VCNEB : calculationMethod

% numSpecies
4
% EndNumSpecies

% atomType
Ar
% EndAtomType

0.0     : ExternalPressure

111     : vcnebType
15      : numImages
500     : numSteps
1       : optimizerType
2       : optReadImages
3       : optRelaxType
0.25    : dt
0.003   : ConvThreshold

0.3     : VarPathLength
3       : K_min
6       : K_max
0       : optFreezing
0       : optMethodCIDI

2       : FormatType
10      : PrintStep

abinitioCode
3
ENDabinit

% commandExecutable
gulp < input > output
% EndExecutable
```

9.4 List of space groups

1	$P1$	2	$P\text{-}1$	3	$P2$	4	$P2_1$
5	$C2\ (A2)^*$	6	Pm	7	$Pc\ (Pa)^*$	8	$Cm\ (Am)^*$
9	$Cc\ (Aa)^*$	10	$P2/m$	11	$P2_1/m$	12	$C2/m\ (A2/m)^*$
13	$P2/c\ (P2/a)^*$	14	$P2_1/c\ (P2_1/a)^*$	15	$C2/c\ (A2/a)^*$	16	$P222$
17	$P222_1$	18	$P2_12_12$	19	$P2_12_12_1$	20	$C222_1$
21	$C222$	22	$F222$	23	$I222$	24	$I2_12_12_1$
25	$Pmm2$	26	$Pmc2_1$	27	$Pcc2$	28	$Pma2$
29	$Pca2_1$	30	$Pnc2$	31	$Pmn2_1$	32	$Pba2$
33	$Pna2_1$	34	$Pnn2$	35	$Cmm2$	36	$Cmc2_1$
37	$Ccc2$	38	$Amm2\ (C2mm)^*$	39	$Aem2\ (C2mb)^*$	40	$Ama2\ (C2cm)^*$
41	$Aea2\ (C2cb)^*$	42	$Fmm2$	43	$Fdd2$	44	$Imm2$
45	$Iba2$	46	$Ima2$	47	$Pmmm$	48	$Pnnn$
49	$Pccm$	50	$Pban$	51	$Pmma$	52	$Pnna$
53	$Pmna$	54	$Pcca$	55	$Pbam$	56	$Pccn$
57	$Pbcm$	58	$Pnnm$	59	$Pmmn$	60	$Pbcn$
61	$Pbca$	62	$Pnma$	63	$Cmcm$	64	$Cmce\ (Cmca)^*$
65	$Cmmm$	66	$Cccm$	67	$Cmme\ (Cmma)^*$	68	$Ccce\ (Ccca)^*$
69	$Fmmm$	70	$Fddd$	71	$Immm$	72	$Ibam$
73	$Ibca$	74	$Imma$	75	$P4$	76	$P4_1$
77	$P4_2$	78	$P4_3$	79	$I4$	80	$I4_1$
81	$P\text{-}4$	82	$I\text{-}4$	83	$P4/m$	84	$P4_2/m$
85	$P4/n$	86	$P4_2/n$	87	$I4/m$	88	$I4_1/a$
89	$P422$	90	$P42_12$	91	$P4_122$	92	$P4_12_12$
93	$P4_222$	94	$P4_22_12$	95	$P4_322$	96	$P4_32_12$
97	$I422$	98	$I4_122$	99	$P4mm$	100	$P4bm$
101	$P4_2cm$	102	$P4_2nm$	103	$P4cc$	104	$P4nc$
105	$P4_2mc$	106	$P4_2bc$	107	$I4mm$	108	$I4cm$
109	$I4_1md$	110	$I4_1cd$	111	$P\text{-}42m$	112	$P\text{-}42c$
113	$P\text{-}42_1m$	114	$P\text{-}42_1c$	115	$P\text{-}4m2$	116	$P\text{-}4c2$
117	$P\text{-}4b2$	118	$P\text{-}4n2$	119	$I\text{-}4m2$	120	$I\text{-}4c2$
121	$I\text{-}42m$	122	$I\text{-}42d$	123	$P4/mmm$	124	$P4/mcc$
125	$P4/nbm$	126	$P4/nnc$	127	$P4/mbm$	128	$P4/mnc$
129	$P4/nmm$	130	$P4/ncc$	131	$P4_2/mmc$	132	$P4_2/mcm$
133	$P4_2/nbc$	134	$P4_2/nnm$	135	$P4_2/mbc$	136	$P4_2/mnm$
137	$P4_2/nmc$	138	$P4_2/ncm$	139	$I4/mmm$	140	$I4/mcm$
141	$I4_1/amd$	142	$I4_1/acd$	143	$P3$	144	$P3_1$
145	$P3_2$	146	$R3$	147	$P\text{-}3$	148	$R\text{-}3$
149	$P312$	150	$P321$	151	$P3_112$	152	$P3_121$
153	$P3_212$	154	$P3_221$	155	$R32$	156	$P3m1$
157	$P31m$	158	$P3c1$	159	$P31c$	160	$R3m$
161	$R3c$	162	$P\text{-}31m$	163	$P\text{-}31c$	164	$P\text{-}3m1$
165	$P\text{-}3c1$	166	$R\text{-}3m$	167	$R\text{-}3c$	168	$P6$
169	$P6_1$	170	$P6_5$	171	$P6_2$	172	$P6_4$
173	$P6_3$	174	$P\text{-}6$	175	$P6/m$	176	$P6_3/m$
177	$P622$	178	$P6_122$	179	$P6_522$	180	$P6_222$
181	$P6_422$	182	$P6_322$	183	$P6mm$	184	$P6cc$
185	$P6_3cm$	186	$P6_3mc$	187	$P\text{-}6m2$	188	$P\text{-}6c2$
189	$P\text{-}62m$	190	$P\text{-}62c$	191	$P6/mmm$	192	$P6/mcc$
193	$P6_3/mcm$	194	$P6_3/mmc$	195	$P23$	196	$F23$
197	$I23$	198	$P2_13$	199	$I2_13$	200	$Pm\text{-}3$
201	$Pn\text{-}3$	202	$Fm\text{-}3$	203	$Fd\text{-}3$	204	$Im\text{-}3$
205	$Pa\text{-}3$	206	$Ia\text{-}3$	207	$P432$	208	$P4_232$
209	$F432$	210	$F4_132$	211	$I432$	212	$P4_332$
213	$P4_132$	214	$I4_132$	215	$P\text{-}43m$	216	$F\text{-}43m$
217	$I\text{-}43m$	218	$P\text{-}43n$	219	$F\text{-}43c$	220	$I\text{-}43d$
221	$Pm\text{-}3m$	222	$Pn\text{-}3n$	223	$Pm\text{-}3n$	224	$Pn\text{-}3m$
225	$Fm\text{-}3m$	226	$Fm\text{-}3c$	227	$Fd\text{-}3m$	228	$Fd\text{-}3c$
229	$Im\text{-}3m$	230	$Ia\text{-}3d$				

*In parentheses there non-standard space groups used in the code.

9.5 List of layer groups

Triclinic							
1	$p1$	2	p-1				
Monoclinic / inclined							
3	$p112$	4	$p11m$	5	$p11a$	6	$p112/m$
7	$p112/a$						
Triclinic / orthogonal							
8	$p211$	9	$p2_111$	10	$c211$	11	$pm11$
12	$pb11$	13	$cm11$	14	$p2/m11$	15	$p2_1/m11$
16	$p2/b11$	17	$p2_1/b11$	18	$c2/m11$		
Orthorhombic							
19	$p222$	20	$p2_122$	21	$p2_12_12$	22	$c222$
23	$pmm2$	24	$pma2$	25	$pba2$	26	$cmm2$
27	$pm2m$	28	$pm2_1b$	29	$pb2_1m$	30	$pb2b$
31	$pm2a$	32	$pm2_1a$	33	$pb2_1a$	34	$pb2n$
35	$cm2m$	36	$cm2e$	37	$pmmm$	38	$pmaa$
39	$pban$	40	$pmam$	41	$pmma$	42	$pman$
43	$pbaa$	44	$pbam$	45	$pbma$	46	$pmmn$
47	$cmmm$	48	$cmme$				
Tetragonal							
49	$p4$	50	p-4	51	$p4/m$	52	$p4/n$
53	$p422$	54	$p42_12$	55	$p4mm$	56	$p4bn$
57	p-$4m2$	58	p-42_1m	59	p-$4m2$	60	p-$4b2$
61	$p4/mmm$	62	$p4/nbm$	63	$p4/mbn$	64	$p4/nmm$
Trigonal							
65	$p3$	66	p-3	67	$p312$	68	$p321$
69	$p3m1$	70	$p31m$	71	p-$31m$	72	p-$3m1$
Hexagonal							
73	$p6$	74	p-6	75	$p6/m$	76	$p622$
77	$p6mm$	78	p-$m2$	79	p-$62m$	80	$p6/mmm$

9.6 List of plane groups

Number	Group
1	p1
2	p2
3	pm
4	pg
5	cm
6	pmm
7	pmg
8	pgg
9	cmm
10	p4
11	p4m
12	p4g
13	p3
14	p3m1
15	p31m
16	p6
17	p6m

9.7 List of point groups

List of all crystallographic and the most important non-crystallographic point groups in Schönflies and Hermann-Mauguin (international) notations.

Crystallographic point groups:

Hermann-Maugin	Schönflies	In USPEX
1	C_1	C1 or E
2	C_2	C2
222	D_2	D2
4	C_4	C4
3	C_3	C3
6	C_6	C6
23	T	T
$\overline{1}$	S_2	S2
M	C_{1h}	Ch1
mm2	C_{2v}	Cv2
$\overline{2}$	S_4	S4
$\overline{3}$	S_6	S6
$\overline{6}$	C_{3h}	Ch3
m$\overline{3}$	T_h	Th
2/m	C_{2h}	Ch2
mmm	D_{2h}	Dh2
4/m	C_{4h}	Ch4
32	D_3	D3
6/m	C_{6h}	Ch6
432	O	O
422	D_4	D4
3m	C_{3v}	Cv3
622	D_6	D6
$\overline{4}$3m	T_d	Td
4mm	C_{4v}	Cv4
$\overline{3}$m	D_{3d}	Dd3
6mm	C_{6v}	Cv6
m$\overline{3}$m	O_h	Oh
$\overline{4}$2m	D_{2d}	Dd2
$\overline{6}$2m	D_{3h}	Dh3
4/mmm	D_{4h}	Dh4
6/mmm	D_{6h}	Dh6
m$\overline{3}$m	O_h	Oh

Important non-crystallographic point groups

Hermann-Maugin	Schönflies	In USPEX
5	C_5	C5
5/m	S_5	S5
$\overline{5}$	S_{10}	S10
5m	Cv_{5v}	Cv5
$\overline{10}$	Ch_{5h}	Ch5
52	D_5	D5
$\overline{5}$m	D_{5d}	Dd5
$\overline{10}$2m	D_{5h}	Dh5
532	I	I
5$\overline{3}$m	I_h	Ih

9.8 Table of univalent covalent radii used in USPEX

Table of covalent radii (in Å) used in USPEX (for hardness calculations, *etc.*):

Z	Element	radius	Z	Element	radius	Z	Element	radius
1	H	0.31	30	Zn	1.22	63	Eu	1.98
2	He	0.28	31	Ga	1.22	64	Gd	1.96
3	Li	1.28	32	Ge	1.20	65	Tb	1.94
4	Be	0.96	33	As	1.19	66	Dy	1.92
5	B	0.84	34	Se	1.20	67	Ho	1.92
6	Csp^3	0.76	35	Br	1.20	68	Er	1.89
	Csp^2	0.73	36	Kr	1.16	69	Tm	1.90
	Csp	0.69	37	Rb	2.20	70	Yb	1.87
7	N	0.71	38	Sr	1.95	71	Lu	1.87
8	O	0.66	39	Y	1.90	72	Hf	1.75
9	F	0.57	40	Zr	1.75	73	Ta	1.70
10	Ne	0.58	41	Nb	1.64	74	W	1.62
11	Na	1.66	42	Mo	1.54	75	Re	1.51
12	Mg	1.41	43	Tc	1.47	76	Os	1.44
13	Al	1.21	44	Ru	1.46	77	Ir	1.41
14	Si	1.11	45	Rh	1.42	78	Pt	1.36
15	P	1.07	46	Pd	1.39	79	Au	1.36
16	S	1.05	47	Ag	1.45	80	Hg	1.32
17	Cl	1.02	48	Cd	1.44	81	Tl	1.45
18	Ar	1.06	49	In	1.42	82	Pb	1.46
19	K	2.03	50	Sn	1.39	83	Bi	1.48
20	Ca	1.76	51	Sb	1.39	84	Po	1.40
21	Sc	1.70	52	Te	1.38	85	At	1.50
22	Ti	1.60	53	I	1.39	86	Rn	1.50
23	V	1.53	54	Xe	1.40	87	Fr	2.60
24	Cr	1.39	55	Cs	2.44	88	Ra	2.21
25	Mn l.s.	1.39	56	Ba	2.15	89	Ac	2.15
	h.s	1.61	57	La	2.07	90	Th	2.06
26	Fe l.s	1.32	58	Ce	2.04	91	Pa	2.00
	h.s.	1.52	59	Pr	2.03	92	U	1.96
27	Co l.s.	1.26	60	Nd	2.01	93	Np	1.90
	h.s.	1.50	61	Pm	1.99	94	Pu	1.87
28	Ni	1.24	62	Sm	1.98	95	Am	1.80
29	Cu	1.32				96	Cm	1.69

Source: Cordero *et al.*, Dalton Trans. 2832-2838, 2008[36].

9.9 Table of default chemical `valences` used in USPEX

Table of chemical `valences` used in USPEX (for hardness calculations, *etc.*):

Z	Element	valence	Z	Element	valence	Z	Element	valence
1	H	1	35	Br	1	69	Tm	3
2	He	0.5	36	Kr	0.5	70	Yb	3
3	Li	1	37	Rb	1	71	Lu	3
4	Be	2	38	Sr	2	72	Hf	4
5	B	3	39	Y	3	73	Ta	5
6	C	4	40	Zr	4	74	W	4
7	N	3	41	Nb	5	75	Re	4
8	O	2	42	Mo	4	76	Os	4
9	F	1	43	Tc	4	77	Ir	4
10	Ne	0.5	44	Ru	4	78	Pt	4
11	Na	1	45	Rh	4	79	Au	1
12	Mg	2	46	Pd	4	80	Hg	2
13	Al	3	47	Ag	1	81	Tl	3
14	Si	4	48	Cd	2	82	Pb	4
15	P	3	49	In	3	83	Bi	3
16	S	2	50	Sn	4	84	Po	2
17	Cl	1	51	Sb	3	85	At	1
18	Ar	0.5	52	Te	2	86	Rn	0.5
19	K	1	53	I	1	87	Fr	1
20	Ca	2	54	Xe	0.5	88	Ra	2
21	Sc	3	55	Cs	1	89	Ac	3
22	Ti	4	56	Ba	2	90	Th	4
23	V	4	57	La	3	91	Pa	4
24	Cr	3	58	Ce	4	92	U	4
25	Mn	4	59	Pr	3	93	Np	4
26	Fe	3	60	Nd	3	94	Pu	4
27	Co	3	61	Pm	3	95	Am	4
28	Ni	2	62	Sm	3	96	Cm	4
29	Cu	2	63	Eu	3	97	Bk	4
30	Zn	2	64	Gd	3	98	Cf	4
31	Ga	3	65	Tb	3	99	Es	4
32	Ge	4	66	Dy	3	100	FM	4
33	As	3	67	Ho	3	101	Md	4
34	Se	2	68	Er	3	102	No	4

9.10 Table of default `goodBonds` used in USPEX

Table of default `goodBonds` used in USPEX (for hardness calculations, *etc.*):

Z	Element	goodBonds	Z	Element	goodBonds	Z	Element	goodBonds
1	H	0.20	35	Br	0.10	69	Tm	0.20
2	He	0.05	36	Kr	0.05	70	Yb	0.20
3	Li	0.10	37	Rb	0.05	71	Lu	0.20
4	Be	0.20	38	Sr	0.10	72	Hf	0.30
5	B	0.30	39	Y	0.20	73	Ta	0.40
6	C	0.50	40	Zr	0.30	74	W	0.30
7	N	0.50	41	Nb	0.35	75	Re	0.30
8	O	0.30	42	Mo	0.30	76	Os	0.30
9	F	0.10	43	Tc	0.30	77	Ir	0.30
10	Ne	0.05	44	Ru	0.30	78	Pt	0.30
11	Na	0.05	45	Rh	0.30	79	Au	0.05
12	Mg	0.10	46	Pd	0.30	80	Hg	0.10
13	Al	0.20	47	Ag	0.05	81	Tl	0.20
14	Si	0.30	48	Cd	0.10	82	Pb	0.30
15	P	0.30	49	In	0.20	83	Bi	0.20
16	S	0.20	50	Sn	0.30	84	Po	0.20
17	Cl	0.10	51	Sb	0.20	85	At	0.10
18	Ar	0.05	52	Te	0.20	86	Rn	0.05
19	K	0.05	53	I	0.10	87	Fr	0.05
20	Ca	0.10	54	Xe	0.05	88	Ra	0.10
21	Sc	0.20	55	Cs	0.05	89	Ac	0.20
22	Ti	0.30	56	Ba	0.10	90	Th	0.30
23	V	0.30	57	La	0.20	91	Pa	0.30
24	Cr	0.25	58	Ce	0.30	92	U	0.30
25	Mn	0.30	59	Pr	0.20	93	Np	0.30
26	Fe	0.25	60	Nd	0.20	94	Pu	0.30
27	Co	0.25	61	Pm	0.20	95	Am	0.30
28	Ni	0.15	62	Sm	0.20	96	Cm	0.30
29	Cu	0.10	63	Eu	0.20	97	Bk	0.30
30	Zn	0.10	64	Gd	0.20	98	Cf	0.30
31	Ga	0.25	65	Tb	0.20	99	Es	0.30
32	Ge	0.50	66	Dy	0.20	100	FM	0.30
33	As	0.35	67	Ho	0.20	101	Md	0.30
34	Se	0.20	68	Er	0.20	102	No	0.30

Bibliography

[1] J. Maddox. Crystals from first principles. *Nature*, 335:201, 1988.

[2] A.R. Oganov and C.W. Glass. Crystal structure prediction using ab initio evolutionary techniques: Principles and applications. *The Journal of Chemical Physics*, 124:244704, 2006.

[3] C.W. Glass, A.R. Oganov, and N. Hansen. USPEX — evolutionary crystal structure prediction. *Comp. Phys. Comm.*, 175:713–720, 2006.

[4] A.R. Oganov and S. Ono. Theoretical and experimental evidence for a post-perovskite phase of MgSiO3 in Earth's D" layer. *Nature*, 430(6998):445–448, July 2004.

[5] M. Murakami, K. Hirose, K. Kawamura, N. Sata, and Y. Ohishi. Post-perovskite phase transition in MgSiO3. *Science*, 304(5672):855–858, 2004.

[6] A.R. Oganov, J.C. Schon, M. Jansen, S.M. Woodley, W.W. Tipton, and R.G. Hennig. *Appendix: First Blind Test of Inorganic Crystal Structure Prediction Methods*, pages 223–231. Wiley-VCH Verlag GmbH & Co. KGaA, 2010.

[7] C.J. Pickard and R.J. Needs. High-pressure phases of silane. *Phys. Rev. Lett.*, 97:045504, Jul 2006.

[8] M. Martinez-Canales, A.R. Oganov, Y. Ma, Y. Yan, A.O. Lyakhov, and A. Bergara. Novel structures and superconductivity of silane under pressure. *Phys. Rev. Lett.*, 102:087005, Feb 2009.

[9] Y. Ma, A.R. Oganov, Y. Xie, Z. Li, and J. Kotakoski. Novel high pressure structures of polymeric nitrogen. *Phys. Rev. Lett.*, 102:065501, 2009.

[10] C.J. Pickard and R.J. Needs. High-pressure phases of nitrogen. *Phys. Rev. Lett.*, 102:125702, Mar 2009.

[11] G. Gao, A.R. Oganov, P. Li, Z. Li, H. Wang, T. Cui, Y. Ma, A. Bergara, A.O. Lyakhov, T. Iitaka, and G. Zou. High-pressure crystal structures and superconductivity of stannane (SnH4). *Proceedings of the National Academy of Sciences*, 107(4):1317–1320, 2010.

[12] C.J. Pickard and R.J. Needs. Structures at high pressure from random searching. *physica status solidi (b)*, 246(3):536–540, 2009.

[13] A.O. Lyakhov, A.R. Oganov, H.T. Stokes, and Q. Zhu. New developments in evolutionary structure prediction algorithm USPEX. *Comp. Phys. Comm.*, 184:1172–1182, 2013.

[14] G.R. Qian, X. Dong, X.-F. Zhou, Y. Tian, A.R. Oganov, and H.-T. Wang. Variable cell nudged elastic band method for studying solid-solid structural phase transitions. *Computer Physics Communications*, 184(9):2111–2118, 2013.

[15] C. Dellago, P.G. Bolhuis, F.S. Csajka, and D. Chandler. Transition path sampling and the calculation of rate constants. *The Journal of Chemical Physics*, 108(5):1964–1977, 1998.

[16] S.E. Boulfelfel, A.R. Oganov, and S. Leoni. Understanding the nature of ”superhard graphite”'. *Scientific Reports*, 2(471):1–9, 2012.

[17] A.R. Oganov and M. Valle. How to quantify energy landscapes of solids. *The Journal of Chemical Physics*, 130:104504, 2009.

[18] A.R. Oganov, A.O. Lyakhov, and M. Valle. How evolutionary crystal structure prediction works — and why. *Accounts of Chemical Research*, 44(3):227–237, 2011.

[19] X.-Q. Chen, H. Niu, D. Li, and Y. Li. Modeling hardness of polycrystalline materials and bulk metallic glasses. *Intermetallics*, 19(9):1275–1281, 2011.

[20] A.R. Oganov and C.W. Glass. Evolutionary crystal structure prediction as a tool in materials design. *Journal of Physics: Condensed Matter*, 20(6):064210, 2008.

[21] L.S. Dubrovinsky, N.A. Dubrovinskaia, V. Swamy, J. Muscat, N.M. Harrison, R. Ahuja, B. Holm, and B. Johansson. Materials science: The hardest known oxide. *Nature*, 410(6829):653–654, 2001.

[22] A.G. Kvashnin, A.R. Oganov, A.I. Samtsevich, and Z. Allahyari. Computational search for novel hard chromium-based materials. *The Journal of Physical Chemistry Letters*, 8(4):755–764, 2017.

[23] A.O. Lyakhov, A.R. Oganov, and M. Valle. *Crystal Structure Prediction Using Evolutionary Approach*, pages 147–180. Wiley-VCH Verlag GmbH & Co. KGaA, 2010.

[24] R. Martoňák, A. Laio, M. Bernasconi, C. Ceriani, P. Raiteri, F. Zipoli, and M. Parrinello. Simulation of structural phase transitions by metadynamics. *Z. Krist.*, 220:489–498, 2005.

[25] W. Zhang, A.R. Oganov, A.F. Goncharov, Q. Zhu, S.E. Boulfelfel, A.O. Lyakhov, E. Stavrou, M. Somayazulu, V.B. Prakapenka, and Z. Konopkova. Unexpected stable stoichiometries of sodium chlorides. *Science*, 342(6165):1502–1505, 2013.

[26] Georg KH Madsen and David J Singh. Boltztrap. a code for calculating band-structure dependent quantities. *Comput. Phys. Commun.*, 175(1):67–71, Jul 2006.

[27] Q. Zhu, L. Li, A.R. Oganov, and P.B. Allen. Evolutionary method for predicting surface reconstructions with variable stoichiometry. *Phys. Rev. B*, 87:195317, May 2013.

[28] S.T. Call, D.Yu. Zubarev, and A.I. Boldyrev. Global minimum structure searches via particle swarm optimization. *Journal of Computational Chemistry*, 28(7):1177–1186, 2007.

[29] Y. Wang, J. Lv, L. Zhu, and Y. Ma. Crystal structure prediction via particle-swarm optimization. *Phys. Rev. B*, 82:094116, Sep 2010.

[30] A.O. Lyakhov, A.R. Oganov, and M. Valle. How to predict very large and complex crystal structures. *Comp. Phys. Comm.*, 181:1623–1632, 2010.

[31] G. Mills, H. Jónsson, and G.K. Schenter. Reversible work transition state theory: application to dissociative adsorption of hydrogen. *Surf. Sci.*, 324(2):305–337, 1995.

[32] G. Henkelman, B.P. Uberuaga, and H. Jónsson. A climbing image nudged elastic band method for finding saddle points and minimum energy paths. *J. Chem. Phys.*, 113(22):9901–9904, 2000.

[33] G. Henkelman and H. Jónsson. Improved tangent estimate in the nudged elastic band method for finding minimum energy paths and saddle points. *J. Chem. Phys.*, 113(22):9978–9985, 2000.

[34] E. Bitzek, P. Koskinen, F. Gähler, M. Moseler, and P. Gumbsch. Structural relaxation made simple. *Phys. Rev. Lett.*, 97(17):170201, 2006.

[35] M. Valle. STM3: a chemistry visualization platform. *Z. Krist.*, 220:585–588, 2005.

[36] B. Cordero, V. Gomez, A.E. Platero-Prats, M. Reves, J. Echeverria, E. Cremades, F. Barragan, and S. Alvarez. Covalent radii revisited. *Dalton Trans.*, 21:2832–2838, 2008.