
Universal Structure Predictor:
Evolutionary Xtallography

A.R. Oganov, S.V. Lepeshkin, S.V. Bakanov, D.V. Rybkovskiy,
D.O. Poletaev, V.N. Valmispild, C.W. Glass, A.O. Lyakhov, Q. Zhu,
Z. Allahyari, G.-R. Qia, E. Trukhan, E.E. Vaneeva

with contributions from

H.T. Stokes, P. Bushlanov, P. Graf, V. Stevanovic, F. Therrien,
A.I. Samtsevich, V.N. Baturin, E. Mazhnik, M.S. Rakin, D. Dong *and
others*

MANUAL

Version 25.0, November 9, 2025.

© A.R. Oganov, S.V. Lepeshkin and S.V. Bakanov

<https://uspex-team.org>

Contents

1	Features, aims and history of USPEX	2
1.1	Overview	2
1.2	Features of USPEX	6
1.3	Key USPEX papers	7
1.4	Version history	8
2	Getting started	13
2.1	How to obtain USPEX	13
2.2	Necessary citations	13
2.3	Bug reports	13
2.4	On which machines USPEX can be run	14
2.5	Codes that can work with USPEX	14
2.6	How to install USPEX	14
2.7	How to run USPEX	14
2.8	USPEX example with MatterSim: a mini-tutorial	15
3	Overview of input and output files	17
3.1	Input files	17
3.1.1	Specific/ folder	18
3.2	Output files	20
4	Input options. The INPUT.txt file	25
4.1	Type of calculation and system specification	25
4.2	Population	29
4.3	Survival of the fittest and selection	30
4.4	Structure generation and variation operators	31
4.5	Constraints	34
4.6	Cell	36
4.7	Restart	38
4.8	Details of <i>ab initio</i> calculations	38
4.9	Fingerprint settings	41
4.10	Space group determination	42

4.11	Keywords for developers	43
4.12	Seldom used keywords	44
4.13	Variable-composition searches: predicting novel compounds and their structures	47
5	Online utilities	50
5.1	Structure characterization	50
5.2	Properties calculations	50
6	Frequently asked questions	51
6.1	How can I visualize the results?	51
6.2	How can I avoid trapping?	51
6.3	What is a single-block calculation?	52
6.4	How to predict structures based on known fragments?	52
6.5	How do I use the seed technique?	53
6.6	How do I play with the compositions?	55
6.7	How do I set up a passwordless connection from a local machine to a remote cluster?	56
6.8	How do I set up a calculation using a job submission script?	57
6.8.1	Step 1: Configuring files in <code>Submission/</code> folder	57
7	Appendices	63
7.1	Sample <code>INPUT.txt</code> files	63
7.1.1	Fixed-composition USPEX calculation (<code>calculationType=300</code>):	63
7.1.2	Variable-composition USPEX calculation (<code>calculationType=301</code>):	64
7.2	List of space groups	65
7.3	List of layer groups	66
7.4	List of plane groups	67
7.5	List of point groups	68
7.6	Table of univalent covalent radii used in USPEX	69
7.7	Table of default chemical valences used in USPEX	70
7.8	Table of default goodBonds used in USPEX	71
	Bibliography	72

1 Features, aims and history of USPEX

1.1 Overview

USPEX stands for *Universal Structure Predictor: Evolutionary Xtallography*... and in Russian “uspek” 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 Maddox¹ 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:

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

If δ is chosen to be a significant fraction of the characteristic bond length (e.g., $\delta = 1 \text{ \AA}$), 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 $\sim 10 \text{ \AA}^3$, 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 ~ 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, Q. Zhu, G.R. Qian, P. Bushlanov, Z. Allahyari, S. Lepeshkin and A. Samtsevich. 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.

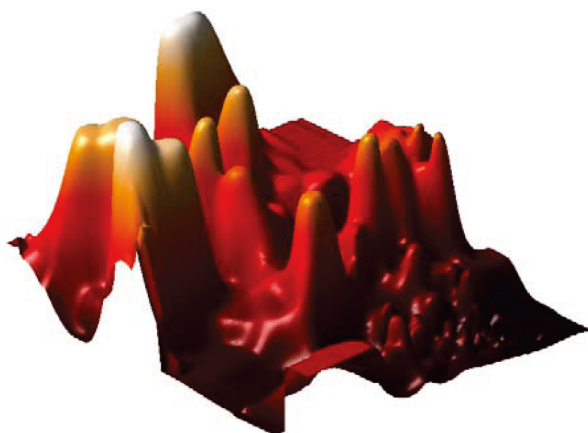


Figure 1: **2D projection of the reduced landscape of Au_3Pd_4 , showing clustering of low-energy structures in one region.** The landscape was produced using the method of Oganov & Valle (2009).

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 Prediction⁶, 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 Pickard⁷ under the name AIRSS) is the simplest, but also the least successful and computationally the

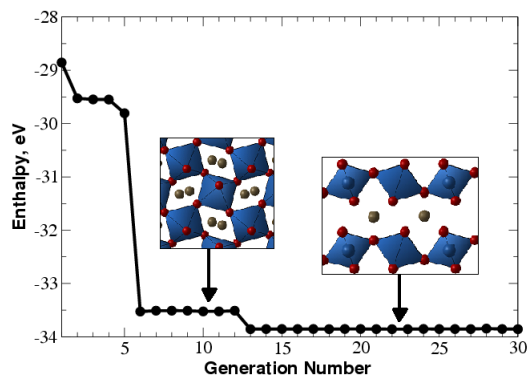


Figure 2: **Prediction of the crystal structure of MgSiO_3 at 120 GPa (20 atoms/cell).** Enthalpy of the best structure as a function of generation is shown. Between the 6th and 12th generations the best structure is perovskite, but at the 13th generation the global minimum (post-perovskite) is found. 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!

most expensive strategy. Even for small systems, such as GaAs with 8 atoms/cell, these 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 ~ 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 stable⁷, turned out to be unstable⁸; and similarly random sampling predictions were shown⁹ to be incorrect for nitrogen¹⁰ and for SnH_4 (compare predictions¹¹ of USPEX and of random sampling¹²).

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 metadynamics¹³, 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

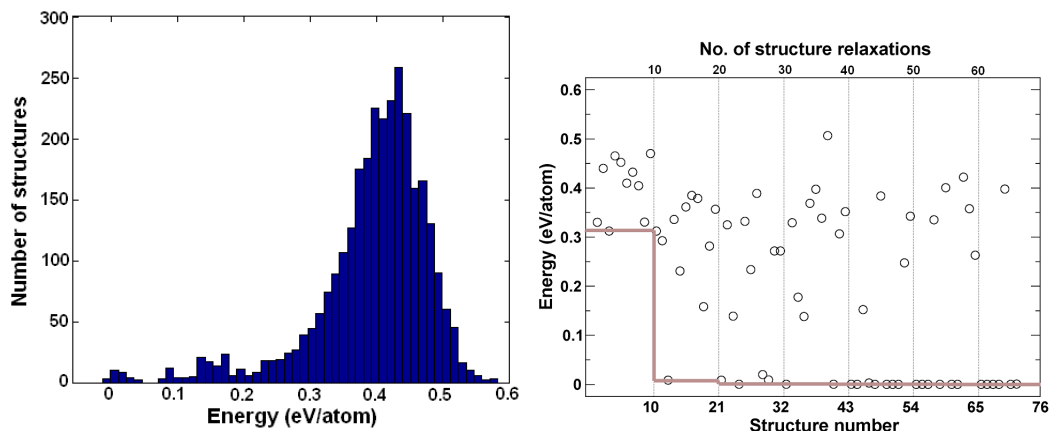


Figure 3: **Structure prediction for GaAs.** a) Energy distribution for relaxed random structures, b) progress of an evolutionary simulation (thin vertical lines show generations of structures, and the grey line shows the lowest energy as a function of generation). All energies are relative to the ground-state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

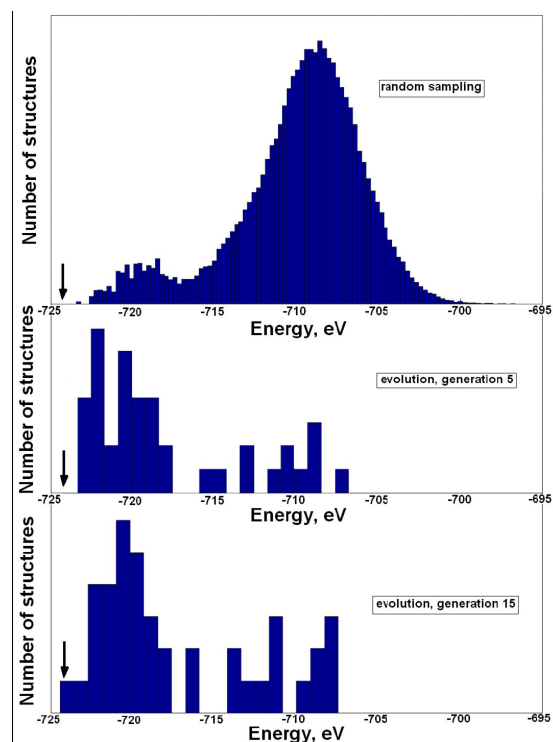


Figure 4: **Sampling of the energy surface: comparison of random sampling and USPEX for a 40-atom cell of MgSiO_3 with cell parameters of post-perovskite.** Energies of locally optimized structures are shown. For random sampling, 1.2×10^5 structures were generated (none of which corresponded to the ground 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.

structures, and even for finding possible phase transition pathways. For detailed investigations of phase transition mechanisms, additional methods are implemented: variable-cell NEB method¹⁴ and transition path method¹⁵ in the version¹⁶.

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 6.5);
 - assembling crystal structures from predefined molecules, including flexible molecules (Subsection ??).
- 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 4.10), calculation of the hardness, order parameters, *etc.*
- Prediction of the structure of nanoparticles and surface reconstructions. See Section ?? 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 6.1).
- USPEX is interfaced with VASP, SIESTA, GULP, LAMMPS, DMACRYS, CP2K, Abinit, CRYSTAL, 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 6.8 for details.

- Options for structure prediction using the USPEX algorithm (default), random sampling, corrected particle swarm optimization (Subsection ??), evolutionary metadynamics (Subsection ??), minima hopping-like algorithm. Capabilities to predict phase transition mechanisms using evolutionary metadynamics, variable-cell NEB method (Subsection ??), and TPS method (Subsection ??). USPEX also has a quick geometric mapping algorithm to predict likely mechanism of a phase transition. (Subsection ??)
- Options to optimize physical properties other than energy — *e.g.*, hardness (Mazhnik & Oganov, 2019), 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 $(\text{H}_2\text{S})_2\text{H}_2$ 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. *Computer Physics Communications*, 236, 1-7.
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. 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.
13. Allahyari, Z., Oganov, A. R. (2018). Multi-objective optimization as a tool for material design. *Handbook of Materials Modeling*, 1-15.

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 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 order¹⁷. 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.

10.3 - Similar to version 10.2, with some critical fixed-bugs in job submission, and some minor fixed-bugs in the code.

10.4 - Several improvements, bug fixes, new features. Local and remote submission files are improved. VCNEB and PSO codes are improved. Interfaces to QE, DMACRYS, VASP, and DFTB+ are improved. Interfaces to Abinit and CRYSTAL codes are added. Half-metallicity fitness is added. Added Mazhnik-Oganov model for calculation of hardness and fracture toughness. All python scripts are translated to python3 - no more python2 needed for USPEX.

10.5 - minor bug fixes. Added optimization of more than one quantity expressed by formula. Improved interface with DFTB+ and GULP5.2. Added “USPEX -u” feature, which allows one to update the package to the latest version without downloading the entire USPEX distribution. Enabled machine learning calculation of the elastic moduli (their optimization is available as `optType=1201-1207`), using graph convolutional neural network, and added Example35 to show how to use this feature.

2 Getting started

2.1 How to obtain USPEX

To obtain USPEX, download the binary package from the official USPEX website:

<https://uspex-team.org>

Use the executable appropriate for your operating system:

- `uspex` for Linux
- `uspex.exe` for Windows

Run the executable in the directory containing your calculation input files.

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 now be run on both Linux and Windows platforms. It requires only one CPU. Using its remote submission mechanism, USPEX can also connect to remote machines (Linux-based) and perform calculations there.

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/>
- GULP — <http://nanochemistry.curtin.edu.au/gulp/>
- ORCA — <https://orcaforum.cec.mpg.de>

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

In this version, no compilation is required. To install USPEX, simply download the archive containing the binary package for your operating system (Linux or Windows) and extract it to your preferred directory. By default, the installation also includes a deep learning-based atomistic simulation model MatterSim¹⁹ package for built-in structure relaxation.

2.7 How to run USPEX

To run USPEX, you only need to install the USPEX package itself — in this case, a deep learning-based atomistic simulation model MatterSim¹⁹ will be used to optimize the structures. Alternatively, structure optimization can be performed using any of the external codes listed in Subsection 2.5.

To set up your calculation see example below 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 as (if you use VASP) `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 (locally 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 6.8 of this Manual.

Run USPEX directly in the calculation folder:

```
./uspex
```

or, on Windows:

```
uspex.exe
```

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.

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, or if you send your jobs to a remote supercomputer. See the keyword `whichCluster` and Section 6.8 of this Manual.

2.8 USPEX example with MatterSim: a mini-tutorial

Once you have downloaded the USPEX package and installed it, you can run the first USPEX example.

Now, let us start our first USPEX experience:

USPEX now provides the possibility to perform built-in structure relaxation using the **MatterSim** machine-learning potential. To enable this mode, simply set:

```
% abinitioCode  
0 0 0  
% ENDabinit
```

By default, three consecutive MatterSim relaxations are performed.

For local calculations, you can control how USPEX runs relaxations using the `whichCluster` parameter:

- `-1` – local run without job submission.
- `0` – local run without job submission but with strict CPU core control (each relaxation runs on a separate core). This mode is recommended for heavy local calculations (e.g., with MatterSim).

The default value is `0`.

You can also specify the number of parallel relaxations using `numParallelCalcs`. This parameter now works for `whichCluster = 0` as well:

```
numParallelCalcs = 10
```

By default, USPEX automatically detects and uses the number of available CPU cores.

The parameter `coresPerJob` defines the number of cores allocated per relaxation, default is 1:

```
coresPerJob = 1
```

The `sleepSeconds` parameter controls how long USPEX waits between sequential structure relaxations, default is 1:

```
sleepSeconds = 5
```

Additionally, in the `input ase.yaml` file, you can specify which MatterSim model to use:

```
mattersim_tier: 1M
```

or

```
mattersim_tier: 5M
```

The `1M` model (`mattersim-v1.0.0-1M.pth`) is faster but less accurate, while the `5M` model (`mattersim-v1.0.0-5M.pth`) offers higher accuracy at a higher computational cost. By default, the `5M` model is used.

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 Subsection 6.1).

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, ORCA, 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 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.*).

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 volume (`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 (ISM EAR=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:	INCAR_2:	INCAR_3:
PREC=LOW	PREC=NORMAL	PREC=NORMAL
EDIFF=1e-2	EDIFF=1e-3	EDIFF=1e-3
EDIFFG=1e-1	EDIFFG=1e-2	EDIFFG=1e-2
NSW=65	NSW=55	ENCUT=520.0
ISIF=4	ISIF=4	NSW=65
IBRION=2	IBRION=1	ISIF=3
POTIM=0.02	POTIM=0.30	IBRION=2
ISM EAR=1	ISM EAR=1	POTIM=0.02
SIGMA=0.10	SIGMA=0.08	ISM EAR=1
		SIGMA=0.07
INCAR_4:	INCAR_5:	
PREC=NORMAL	PREC=NORMAL	
EDIFF=1e-4	EDIFF=1e-4	
EDIFFG=1e-3	EDIFFG=1e-3	
ENCUT=600.0	ENCUT=600.0	
NSW=55	NSW=0	
ISIF=3	ISIF=2	
IBRION=1	IBRION=2	
POTIM=0.30	POTIM=0.02	
ISM EAR=1	ISM EAR=1	
SIGMA=0.06	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)  (Å³)  (g/cm³)
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  0
 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.
 - `Surface_Diagram.pdf` — file (only for variable-composition surface structure predictions) showing surface phase diagram.

4 Input options. The INPUT.txt file

Typical INPUT.txt files are given in the Appendix 7.1. 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 https://uspex-team.org/online_utilities/ — these help to prepare the INPUT.txt file, molecular files, and analyze some of results. Section 5 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

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
- molecularity:
 - “0” — non-molecular
 - “1” — molecular calculations
- variability of chemical composition in the calculation:
 - “0” — fixed composition
 - “1” — variable composition

Default: 300

Format:

```
301 : calculationType
```

▷ *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)

Default: enthalpy

Format:

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

▷ *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 0
% 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 $x, y = (0,1,2,\dots)$ — 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* `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 7.7). 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.

Format:

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

▷ *variable* `goodBonds`

Meaning: Specifies, 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 $\text{goodBonds} = \frac{\text{valence}}{\text{max_coordination_number}}$ 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 7.8

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* `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). 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. 5).

Default:

- For 3D crystals: `2-230`

Format:

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

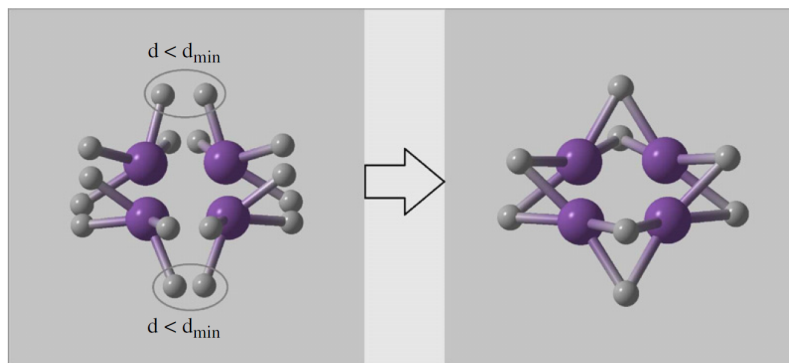


Figure 5: **Example of random symmetric structure generation and merging atoms onto special Wyckoff positions (for detail, see Ref. ¹³).**

▷ *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 Reference¹³).

▷ *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 `fracPyXTal`

Meaning: Specifies the percentage of structures obtained by program PyXTal.

Default: 0.05

Format:

0.05 : `fracPyXTal`

▷ variable `howManySwaps`

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

Default: $0.5 \times (\text{maximum number of possible swaps})$. 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 $\alpha = \beta = \gamma \sim 120^\circ$ are practically flat). Therefore, we developed^{20;21} 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 (**a** + **b**) these criteria are:

$$\left| \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{b}|} \right| > \frac{|\mathbf{b}|}{2} \quad (2)$$

$$\left| \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}|} \right| > \frac{|\mathbf{a}|}{2} \quad (3)$$

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

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

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

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

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: $1.8 \times$ covalent diameter of the largest atom. For molecular crystals (`calculationType = 310, 311`) default value is $1.8 \times \text{max}(\text{MolCenters})$.

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
```

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
```

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

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

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

(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 https://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
```

Alternative Format:

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

The difference is that here one (or more) stage(s) of calculation for each structure is shown in parentheses. This is very useful when optimizing physical properties: for calculations in parentheses, the structure is symmetrized and represented in the standard crystallographic

setting; this allows us to fully use crystal symmetry and make property calculations cheaper and more numerically robust. Before symmetrization and property calculations, the structure must be well relaxed (so the user must make sure that relaxation is well done).

Note 1: the enthalpy is taken from the last stage **BEFORE** parentheses.

Note 2: Numbers indicate the code used at each step of structure relaxation:

0 — MATTERSIM	3 — GULP
1 — VASP	5 — ORCA

▷ *variable* `KresolStart`

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

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 \AA between neighboring clusters in adjacent unit cells). Used only for surfaces, 2D-crystals, and nanoparticles.

Default: 10 \AA 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, 2009¹⁷) 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 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
```

4.11 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 algorithm, 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`

4.12 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;22}, 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: $3 \times (\text{average atomic radius})$

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 from 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

4.13 Variable-composition seraches: predicting novel compounds 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.

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

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.*

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. 6

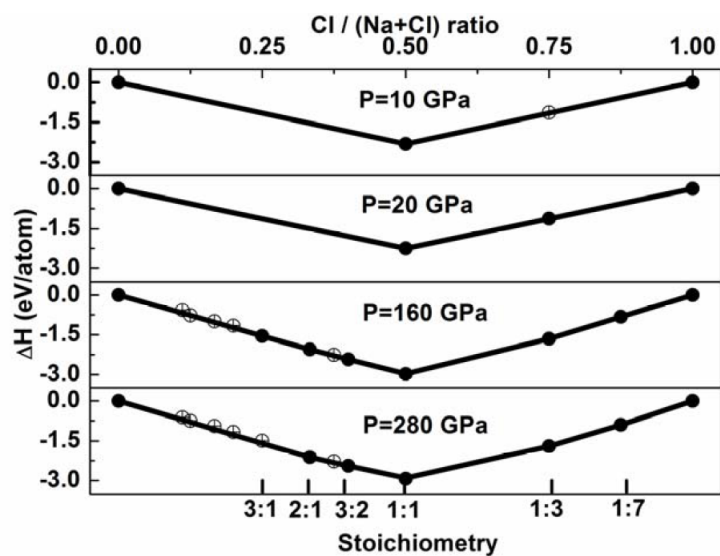


Figure 6: **Convex hull diagram for Na-Cl system at selected pressures.** Solid circles represent stable compounds; open circles — metastable compounds.

show stable sodium chlorides discovered using USPEX and confirmed by experiment²³.

5 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:

https://uspex-team.org/online_utilities/

Below you can find information about each one of them.

5.1 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.

5.2 Properties calculations

Here we have 2 utilities:

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

6 Frequently asked questions

6.1 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 toolkit²⁴, 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. 7 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>.

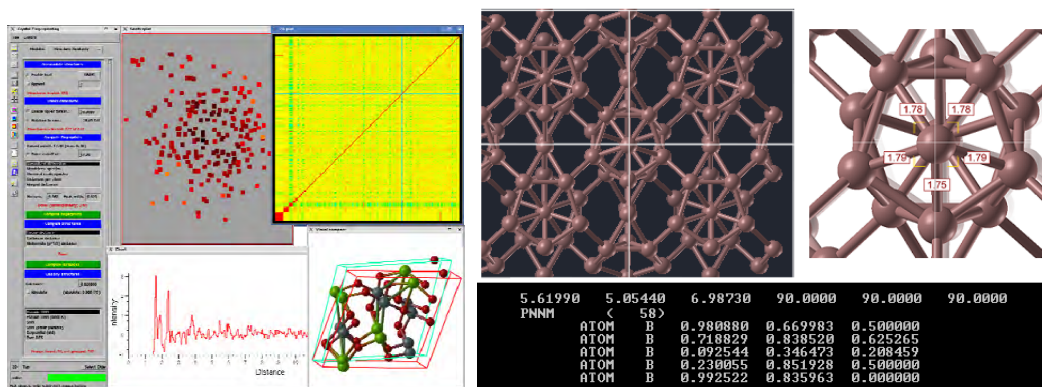


Figure 7: STM4 interface for USPEX.

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

6.2 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.

6.3 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 0
% 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).

6.4 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

% numSpecies
48
% EndNumSpecies
```

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 ???. 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 this 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 fragment feature simultaneously. It is highly recommended to use it with the fixed cell mode, when the cell parameters are available.

6.5 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 $P6_3/mc$ H₂ 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
-0.000000  5.490285  0.000000
0.000000  0.000000  4.508715
H
16
Direct

```

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

6.6 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 C₈, O₈, and C₂O₄ 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.

6.7 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
```

6.8 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 programming 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.

6.8.1 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`:

```

1 from subprocess import check_output
import re
3 import sys

5
def submitJob_local(index : int , commnadExecutable : str) -> int:
7     """
    This routine is to submit job locally
9     One needs to do a little edit based on your own case.

11    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.
13    Step 3: to get the jobID from the screen message
    :return: job ID
15    """

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

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

35    # Step 3
    # Here we parse job ID from the output of previous command
37    jobNumber = int(re.findall(r'\d+', output)[0])
    return jobNumber
39

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

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

```

In checkStatus_local.py:

```

import argparse
2 import glob
import os

4
from subprocess import check_output

6
_author_ = 'etikhonov'

8

```

```

10 def checkStatus_local(jobID : int) -> bool:
11     """
12     This function is to check if the submitted job is done or not
13     One needs to do a little edit based on your own case.
14     1 : whichCluster (0: no-job-script, 1: local submission, 2: remote submission)
15     Step1: the command to check job by ID.
16     Step2: to find the keywords from screen message to determine if the job is done
17     Below is just a sample:
18
19     |-----|
20     | Job id           | Name       | User   | Time Use | S | Queue |
21     |-----|-----|-----|-----|---|-----|
22     | 2455453.nano    | USPEX     | qzhu  | 02:28:42 | R | cfn_gen04 |
23     |-----|-----|-----|-----|---|-----|
24
25     If the job is still running, it will show as above.
26
27     If there is no key words like 'R/Q Cfn_gen04', it indicates the job is done.
28     :param jobID:
29     :return: doneOr
30     """
31
32     # Step 1
33     output = str(check_output('qstat {}'.format(jobID), shell=True))
34     # Step 2
35     doneOr = True
36     if 'R' in output or 'Q' in output:
37         doneOr = False
38     if doneOr:
39         for file in glob.glob('USPEX*'):
40             os.remove(file) # to remove the log file
41     return doneOr
42
43 if __name__ == '__main__':
44     parser = argparse.ArgumentParser()
45     parser.add_argument('-j', dest='jobID', type=int)
46     args = parser.parse_args()
47
48     isDone = checkStatus_local(jobID=args.jobID)
49     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:

```

1 import argparse
2 import os
3 import re
4
5 from subprocess import check_output
6
7
8 def submitJob_remote(workingDir : str, index : int, commandExecutable : str) -> int:
9     """

```

```

10 This routine is to submit job to remote cluster
11 One needs to do a little edit based on your own case.
12 Step 1: to prepare the job script which is required by your supercomputer
13 Step 2: to submit the job with the command like qsub, bsub, llsubmit, .etc.
14 Step 3: to get the jobID from the screen message

16 :param workingDir: working directory on remote machine
17 :param index: index of the structure.
18 :param commandExecutable: command executable for current step of optimization
19 :return:
20 """

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

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

40
41 # Create the remote directory
42 # Please change the ssh/scp command if necessary.
43 try:
44     os.system('ssh -i ~/.ssh/id_rsa ' + Address + ' mkdir -p ' + Path)
45 except:
46     pass

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

54
55 # Step 2
56 # Run command
57 output = str(check_output('ssh -i ~/.ssh/id_rsa ' + Address + ' qsub ' + Path + '/
58 myrun', shell=True))

59
60 # Step 3
61 # Here we parse job ID from the output of previous command
62 jobNumber = int(re.findall(r'\d+', output)[0])
63 return jobNumber

64
65 if __name__ == '__main__':
66     parser = argparse.ArgumentParser()
67     parser.add_argument('-i', dest='index', type=int)
68     parser.add_argument('-c', dest='commandExecutable', type=str)
69     parser.add_argument('-f', dest='workingDir', type=str)
70     args = parser.parse_args()
71
72     jobNumber = submitJob.remote(workingDir=args.workingDir, index=args.index,

```

```
74 | commnadExecutable=args.commnadExecutable)  
    | print('CALLBACK ' + str(jobNumber))
```

In `checkStatus_remote.py`:

```

1 import argparse
  import os
3
4 from subprocess import check_output
5
6 def checkStatus_remote(jobID : int , workingDir : str , index : int) -> bool:
7     """
8     This routine is to check if the submitted job is done or not
9     One needs to do a little edit based on your own case.
10    Step1: Specify the PATH to put your calculation folder
11    Step2: Check JobID, the exact command to check job by jobID
12    :param jobID:
13    :param index:
14    :param workingDir:
15    :return:
16    """
17    # Step 1
18    Home = '/home/etikhonov' # 'pwd' of your home directory of your remote machine
19    Address = 'rurik' # Your target supercomputer: username@address or ssh alias
20    # example of address: user@somedomain.edu -p 2222
21    Path = Home + '/' + workingDir + '/CalcFold' + str(index) # just keep it
22
23    # Step 2
24    output = str(check_output('ssh ' + Address + ' qstat ' + str(jobID), shell=True))
25    # If you using full address without ssh alias , you must provide valid ssh private key
26    # like there:
27    # output = str(check_output('ssh -i ~/.ssh/id_rsa ' + Address + ' /usr/bin/qstat ' +
28    # str(jobID), shell=True))
29
30    if not 'R' in output or not 'Q' in output:
31        doneOr = True
32        # [nothing, nothing] = unix(['scp -i ~/.ssh/id_rsa ' Address ':' Path '/OUTCAR ./
33        ']) %OUTCAR is not necessary by default
34        os.system('scp ' + Address + ':' + Path + '/OSZICAR ./') # For reading enthalpy/
35        energy
36        os.system('scp ' + Address + ':' + Path + '/CONTCAR ./') # For reading
37        structural info
38        # Edit ssh command as above!
39    else:
40        doneOr = False
41    return doneOr
42
43 if __name__ == '__main__':
44     parser = argparse.ArgumentParser()
45     parser.add_argument('-j', dest='jobID', type=int)
46     parser.add_argument('-i', dest='index', type=int)
47     parser.add_argument('-f', dest='workingDir', type=str)
48     args = parser.parse_args()
49
50     isDone = checkStatus_remote(jobID=args.jobID, workingDir=args.workingDir, index=args.
51     index)
52     print('CALLBACK ' + str(int(isDone)))

```

7 Appendices

7.1 Sample INPUT.txt files

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

```
PARAMETERS EVOLUTIONARY ALGORITHM
2 % Example of the short input , using most options as defaults
4 % atomType
Mg Al O
6 % EndAtomType
8 % numSpecies
2 4 8
10 % EndNumSpecies
12 50 : numGenerations
50.0 : ExternalPressure
14
16 % abinitioCode
3 3 3 3 3
% ENDabinit
18
20 % commandExecutable
gulp < input > output
% EndExecutable
```

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

```
1 USPEX : calculationMethod (USPEX, VCNEB, META)
2 301   : calculationType (dimension: 0-3; molecule: 0/1; varcomp: 0/1)
3 1     : AutoFrac

5 % atomType
6 Mo B
7 % EndAtomType

9 % numSpecies
10 1 0
11 0 1
12 % EndNumSpecies

13
14 80    : populationSize
15 200   : initialPopSize
16 60    : numGenerations
17 20    : stopCrit

18
19 11    : firstGeneMax
20 8     : minAt
21 18   : maxAt

22
23 % abinitioCode
24 3 3 3
25 % ENDabinit

26
27 % commandExecutable
28 gulp < input > output
29 % EndExecutable
```

7.2 List of space groups

1	<i>P1</i>	2	<i>P-1</i>	3	<i>P2</i>	4	<i>P2₁</i>
5	<i>C2 (A2)*</i>	6	<i>Pm</i>	7	<i>Pc (Pa)*</i>	8	<i>Cm (Am)*</i>
9	<i>Cc (Aa)*</i>	10	<i>P2/m</i>	11	<i>P2₁/m</i>	12	<i>C2/m (A2/m)*</i>
13	<i>P2/c (P2/a)*</i>	14	<i>P2₁/c (P2₁/a)*</i>	15	<i>C2/c (A2/a)*</i>	16	<i>P222</i>
17	<i>P222₁</i>	18	<i>P2₁2₁2</i>	19	<i>P2₁2₁2₁</i>	20	<i>C222₁</i>
21	<i>C222</i>	22	<i>F222</i>	23	<i>I222</i>	24	<i>I2₁2₁2₁</i>
25	<i>Pmm2</i>	26	<i>Pmc2₁</i>	27	<i>Pcc2</i>	28	<i>Pma2</i>
29	<i>Pca2₁</i>	30	<i>Pnc2</i>	31	<i>Pmn2₁</i>	32	<i>Pba2</i>
33	<i>Pna2₁</i>	34	<i>Pnn2</i>	35	<i>Cmm2</i>	36	<i>Cmc2₁</i>
37	<i>Ccc2</i>	38	<i>Amm2 (C2mm)*</i>	39	<i>Aem2 (C2mb)*</i>	40	<i>Ama2 (C2cm)*</i>
41	<i>Aea2 (C2cb)*</i>	42	<i>Fmm2</i>	43	<i>Fdd2</i>	44	<i>Imm2</i>
45	<i>Iba2</i>	46	<i>Ima2</i>	47	<i>Pmmm</i>	48	<i>Pnnn</i>
49	<i>Pccm</i>	50	<i>Pban</i>	51	<i>Pmma</i>	52	<i>Pnna</i>
53	<i>Pmna</i>	54	<i>Pcca</i>	55	<i>Pbam</i>	56	<i>Pccn</i>
57	<i>Pbcm</i>	58	<i>Pnnm</i>	59	<i>Pmnn</i>	60	<i>Pbcn</i>
61	<i>Pbca</i>	62	<i>Pnma</i>	63	<i>Cmcm</i>	64	<i>Cmce (Cmca)*</i>
65	<i>Cmmm</i>	66	<i>Cccm</i>	67	<i>Cmme (Cmma)*</i>	68	<i>Ccce (Ccca)*</i>
69	<i>Fmmm</i>	70	<i>Fddd</i>	71	<i>Immm</i>	72	<i>Ibam</i>
73	<i>Ibca</i>	74	<i>Imma</i>	75	<i>P4</i>	76	<i>P4₁</i>
77	<i>P4₂</i>	78	<i>P4₃</i>	79	<i>I4</i>	80	<i>I4₁</i>
81	<i>P-4</i>	82	<i>I-4</i>	83	<i>P4/m</i>	84	<i>P4₂/m</i>
85	<i>P4/n</i>	86	<i>P4₂/n</i>	87	<i>I4/m</i>	88	<i>I4₁/a</i>
89	<i>P422</i>	90	<i>P42₁2</i>	91	<i>P4₁22</i>	92	<i>P4₁2₁2</i>
93	<i>P4₂22</i>	94	<i>P4₂2₁2</i>	95	<i>P4₃22</i>	96	<i>P4₃2₁2</i>
97	<i>I422</i>	98	<i>I4₁22</i>	99	<i>P4mm</i>	100	<i>P4bm</i>
101	<i>P4₂cm</i>	102	<i>P4₂nm</i>	103	<i>P4cc</i>	104	<i>P4nc</i>
105	<i>P4₂mc</i>	106	<i>P4₂bc</i>	107	<i>I4mm</i>	108	<i>I4cm</i>
109	<i>I4₁md</i>	110	<i>I4₁cd</i>	111	<i>P-42m</i>	112	<i>P-42c</i>
113	<i>P-42₁m</i>	114	<i>P-42₁c</i>	115	<i>P-4m2</i>	116	<i>P-4c2</i>
117	<i>P-4b2</i>	118	<i>P-4n2</i>	119	<i>I-4m2</i>	120	<i>I-4c2</i>
121	<i>I-42m</i>	122	<i>I-42d</i>	123	<i>P4/mmm</i>	124	<i>P4/mcc</i>
125	<i>P4/nbm</i>	126	<i>P4/nmc</i>	127	<i>P4/mbm</i>	128	<i>P4/mnc</i>
129	<i>P4/nmm</i>	130	<i>P4/ncc</i>	131	<i>P4₂/mcc</i>	132	<i>P4₂/mcm</i>
133	<i>P4₂/nbc</i>	134	<i>P4₂/nmm</i>	135	<i>P4₂/mbc</i>	136	<i>P4₂/mmm</i>
137	<i>P4₂/nmc</i>	138	<i>P4₂/ncm</i>	139	<i>I4/mmm</i>	140	<i>I4/mcm</i>
141	<i>I4₁/amd</i>	142	<i>I4₁/acd</i>	143	<i>P3</i>	144	<i>P3₁</i>
145	<i>P3₂</i>	146	<i>R3</i>	147	<i>P-3</i>	148	<i>R-3</i>
149	<i>P312</i>	150	<i>P321</i>	151	<i>P3₁12</i>	152	<i>P3₁21</i>
153	<i>P3₂12</i>	154	<i>P3₂21</i>	155	<i>R32</i>	156	<i>P3m1</i>
157	<i>P31m</i>	158	<i>P3c1</i>	159	<i>P31c</i>	160	<i>R3m</i>
161	<i>R3c</i>	162	<i>P-31m</i>	163	<i>P-31c</i>	164	<i>P-3m1</i>
165	<i>P-3c1</i>	166	<i>R-3m</i>	167	<i>R-3c</i>	168	<i>P6</i>
169	<i>P6₁</i>	170	<i>P6₅</i>	171	<i>P6₂</i>	172	<i>P6₄</i>
173	<i>P6₃</i>	174	<i>P-6</i>	175	<i>P6/m</i>	176	<i>P6₃/m</i>
177	<i>P622</i>	178	<i>P6₁22</i>	179	<i>P6₅22</i>	180	<i>P6₂22</i>
181	<i>P6₄22</i>	182	<i>P6₃22</i>	183	<i>P6mm</i>	184	<i>P6cc</i>
185	<i>P6₃cm</i>	186	<i>P6₃mc</i>	187	<i>P-6m2</i>	188	<i>P-6c2</i>
189	<i>P-62m</i>	190	<i>P-62c</i>	191	<i>P6/mmm</i>	192	<i>P6/mcc</i>
193	<i>P6₃/mcm</i>	194	<i>P6₃/mmc</i>	195	<i>P23</i>	196	<i>F23</i>
197	<i>I23</i>	198	<i>P2₁3</i>	199	<i>I2₁3</i>	200	<i>Pm-3</i>
201	<i>Pn-3</i>	202	<i>Fm-3</i>	203	<i>Fd-3</i>	204	<i>Im-3</i>
205	<i>Pa-3</i>	206	<i>Ia-3</i>	207	<i>P432</i>	208	<i>P4₂32</i>
209	<i>F432</i>	210	<i>F4₁32</i>	211	<i>I432</i>	212	<i>P4₃32</i>
213	<i>P4₁32</i>	214	<i>I4₁32</i>	215	<i>P-43m</i>	216	<i>F-43m</i>
217	<i>I-43m</i>	218	<i>P-43n</i>	219	<i>F-43c</i>	220	<i>I-43d</i>
221	<i>Pm-3m</i>	222	<i>Pn-3n</i>	223	<i>Pm-3n</i>	224	<i>Pn-3m</i>
225	<i>Fm-3m</i>	226	<i>Fm-3c</i>	227	<i>Fd-3m</i>	228	<i>Fd-3c</i>
229	<i>Im-3m</i>	230	<i>Ia-3d</i>				

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

7.3 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	$p6/mmm$

7.4 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

7.5 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-Mauguin	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
$\bar{1}$	S ₂	S2
M	C _{1h}	Ch1
mm2	C _{2v}	Cv2
$\bar{2}$	S ₄	S4
$\bar{3}$	S ₆	S6
$\bar{6}$	C _{3h}	Ch3
m $\bar{3}$	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
$\bar{4}3m$	T _d	Td
4mm	C _{4v}	Cv4
$\bar{3}m$	D _{3d}	Dd3
6mm	C _{6v}	Cv6
m $\bar{3}m$	O _h	Oh
$\bar{4}2m$	D _{2d}	Dd2
$\bar{6}2m$	D _{3h}	Dh3
4/mmm	D _{4h}	Dh4
6/mmm	D _{6h}	Dh6
m $\bar{3}m$	O _h	Oh

Important non-crystallographic point groups

Hermann-Mauguin	Schönflies	In USPEX
5	C ₅	C5
5/m	S ₅	S5
$\bar{5}$	S ₁₀	S10
5m	Cv _{5v}	Cv5
$\bar{10}$	Ch _{5h}	Ch5
52	D ₅	D5
$\bar{5}m$	D _{5d}	Dd5
$\bar{10}2m$	D _{5h}	Dh5
532	I	I
5 $\bar{3}m$	I _h	Ih

7.6 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, 2008²⁵.

7.7 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

7.8 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] Yichi Zhou Xixian Liu Yu Shi Jielan Li Guanzhi Li Zekun Chen Shuizhou Chen Claudio Zeni Matthew Horton Robert Pinsler Andrew Fowler Daniel Zügner Tian Xie Jake Smith Lixin Sun Qian Wang Lingyu Kong Chang Liu Hongxia Hao Ziheng Lu Han Yang, Chenxi Hu. Mattersim: A deep learning atomistic model across elements, temperatures and pressures. May 2024.
- [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] 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.
- [22] 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.
- [23] 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.
- [24] M. Valle. STM3: a chemistry visualization platform. *Z. Krist.*, 220:585–588, 2005.
- [25] B. Cordero, V. Gomez, A.E. Platero-Prats, M. Reves, J. Echeverria, E. Cremades, F. Baragan, and S. Alvarez. Covalent radii revisited. *Dalton Trans.*, 21:2832–2838, 2008.