Universal Structure Predictor:

Evolutionary Xtallography

Code by **P. Bushlanov, S. Lepeshkin, A.R. Oganov**

based and improved upon earlier version by
A.R. Oganov, Z. Allahyari, C.W. Glass, A.O. Lyakhov, Q. Zhu, G.-R. Qian, H.T. Stokes, V. Stevanovic, S. Lepeshkin, E. Mazhnik, and others

with contributions from
A. Samtsevich, M. Galasso, V. Baturin, D. Rybkovskiy, A. Mazitov.

MANUAL

Version 2023.0, May 23, 2023.

https://uspex-team.org

CONTENTS

CHAPTER

ONE

FEATURES, AIMS AND HISTORY OF USPEX

1.1 Overview

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

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

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

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

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

$$C = \frac{1}{(V/\delta^3)} \frac{(V/\delta^3)!}{[(V/\delta^3)-N]!N!} \tag{1.1.1}$$

If δ is chosen to be a significant fraction of the characteristic bond length (e.g., $\delta = 1$Å 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 ~ 10Å^3, and taking into account Stirling's formula ($n! \approx \sqrt{2\pi n}(n/e)^n$), the number of possible structures for an element A (compound AB) is 10^{11} (10^{14}) for a system with 10 atoms in the unit cell, 10^{25} (10^{30}) for a system with 20 atoms in the cell, and 10^{39} (10^{47}) for a system with 30 atoms in the unit cell. These numbers are enormous and practically impossible to deal with even for small systems with a total number of atoms $N \sim 10$. Even worse, complexity increases exponentially with N. It is clear then, that point-by-point exploration of the free energy surface going through all possible structures is not feasible, except for the simplest systems with $\sim$1-5 atoms in the unit cell.

USPEX [2,3] employs an evolutionary algorithm devised by *A.R. Oganov* and *C.W. Glass*, with major subsequent contributions by *A.O. Lyakhov*, *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. 1.1.1). Furthermore, by carefully designing variation operators, it is very easy to incorporate additional features into an evolutionary algorithm.

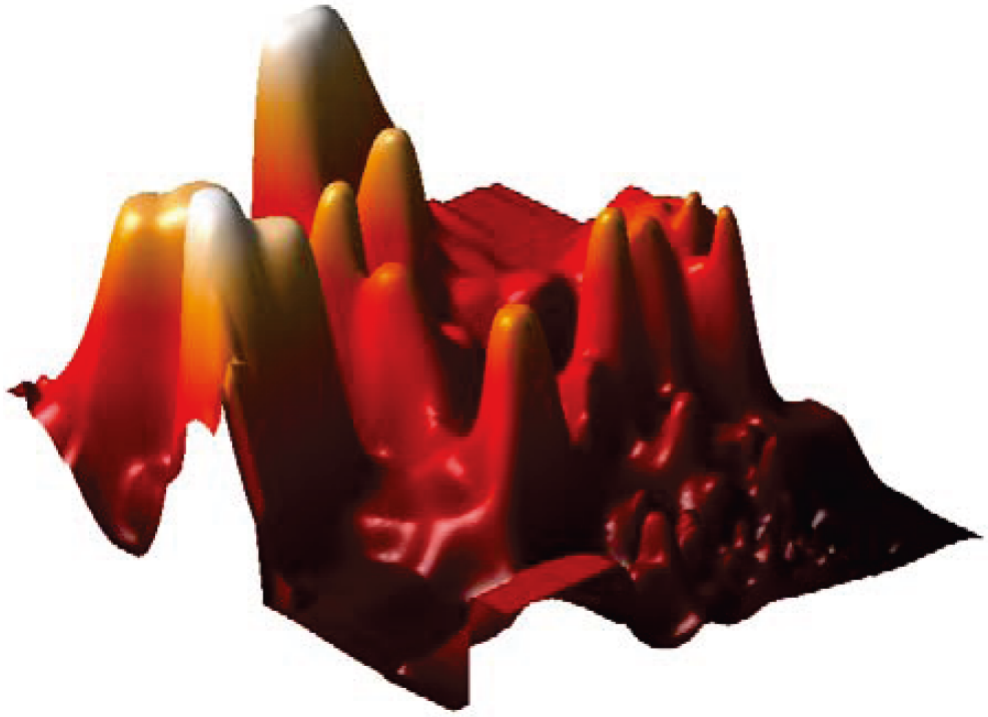

Fig. 1.1.1: 2D projection of the reduced landscape of Au_8Pd_4, showing clustering of low-energy structures in one region. The landscape was produced using the method of Oganov & Valle (2009).

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

The popularity of USPEX is due to its extremely high efficiency and reliability. This was shown in the First Blind Test for Inorganic Crystal Structure Prediction [6], where USPEX outperformed the other methods it was tested against (simulated annealing and random sampling). Random sampling (a technique pioneered for structure prediction by Freeman and Schmidt in 1993 and 1996, respectively, and since 2006 revived by Pickard [7] under the name AIRSS) is the simplest, but also the least successful and computationally the most expensive strategy. Even for small systems, such as GaAs with 8 atoms/cell, these advantages are large (random sampling requires on average 500 structure relaxations to find the ground state in this case, while USPEX finds it after only $\sim$ 30 relaxations! (Fig. 1.1.3)). Due to the exponential scaling of the complexity of structure search (Eq.1.1.1), the advantages of USPEX increase exponentially with system size. For instance, 2 out of 3 structures of SiH_4 predicted by random sampling to be stable [7], turned out to be unstable [8]; and similarly random sampling predictions were shown [9] to be incorrect for nitrogen [10] and for SnH_4 (compare predictions [11] of USPEX and of random sampling [12]).

For larger systems, random sampling tends to produce almost exclusively disordered structures with nearly identical energies, which decreases the success rate to practically zero, as shown in the example of $MgSiO_3$ post-perovskite with 40 atoms/supercell — random sampling fails to find the correct structure even after 120,000 relaxations, whereas USPEX finds it after several hundred relaxations (Fig. 1.1.4).

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 (first developed for predicting structures of clusters by A.I.Boldyrev [13]) has been revamped and implemented on the basis of USPEX with minor programming work as a corrected PSO (corPSO) algorithm, which outperformed previous versions of PSO. Still, any

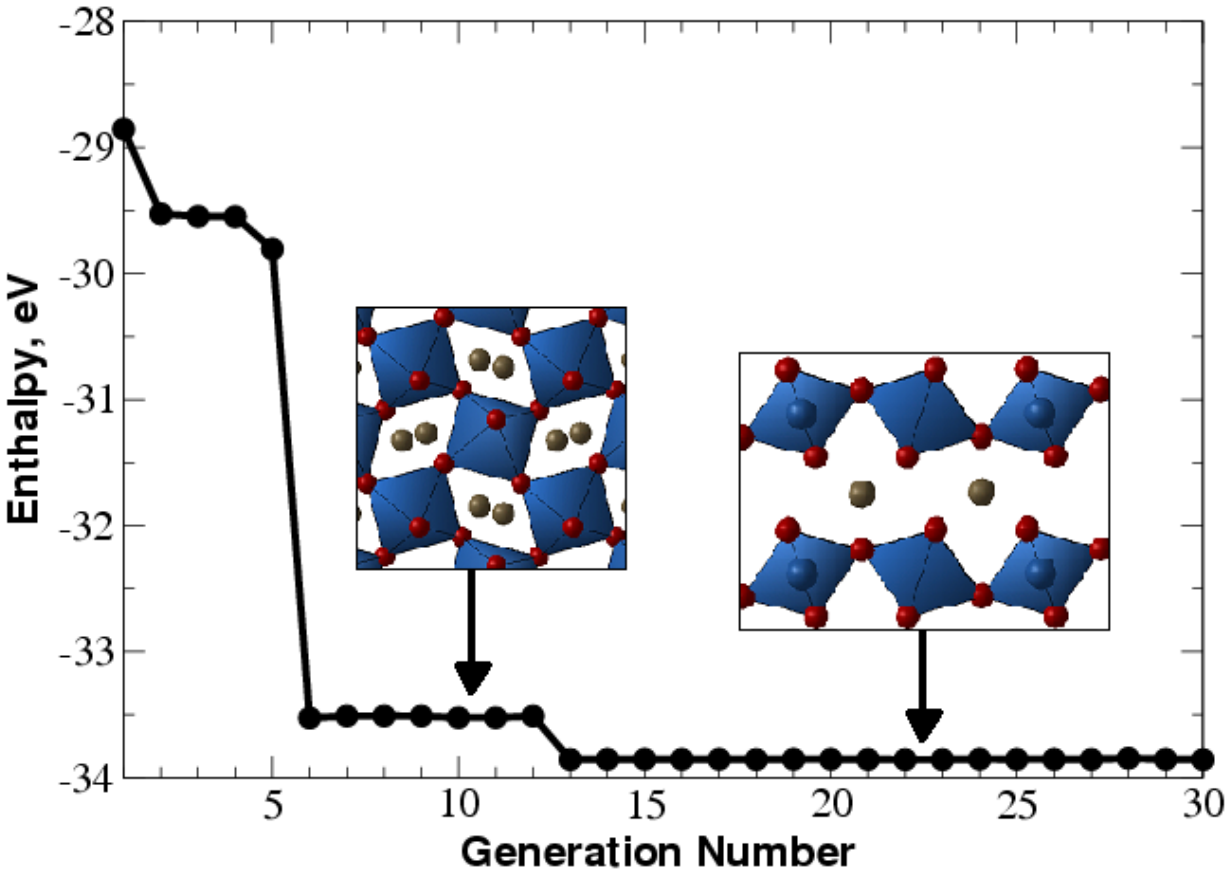

Fig. 1.1.2: Prediction of the crystal structure of $MgSiO_3$ at 120GPa (20 atoms/cell). Enthalpy of the best structure as a function of generation is shown. Between the 6^{th} and 12^{th} generations the best structure is perovskite, but at the 13^{th} generation the global minimum (post-perovskite) 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 $\sim$10 calculations performed by the very first version of USPEX — and even that was pretty fast!

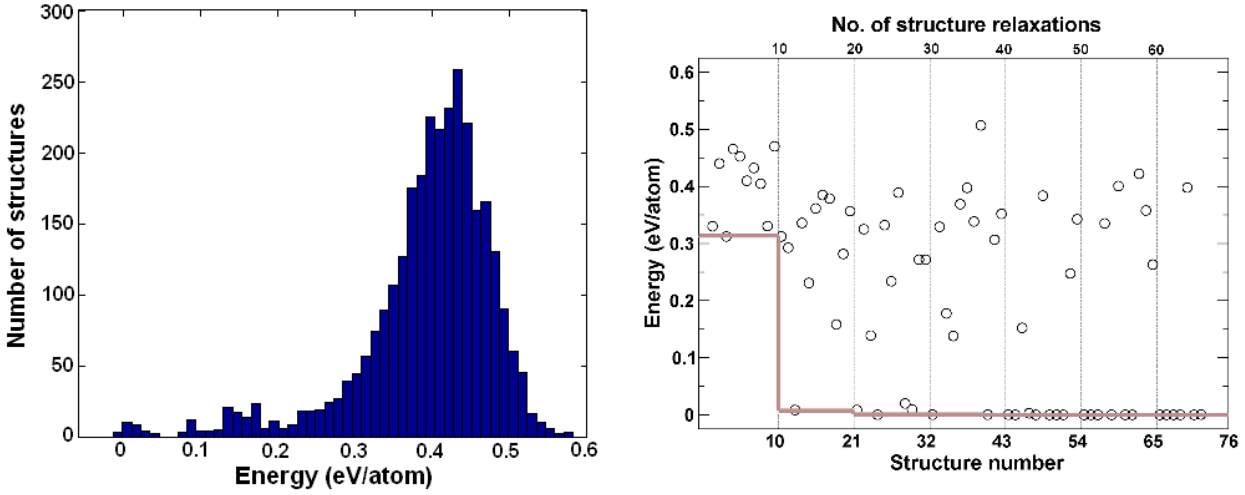

Fig. 1.1.3: a) Energy distribution for relaxed random structures, b) progress of an evolutionary simulation (thin vertical lines show generations of structures, and the grey line shows the lowest energy as a function of generation). All energies are relative to the ground-state structure. The evolutionary simulation used 10 structures per generation. In addition, the lowest-energy structure of the previous generation survived into the next generation.

version of PSO is rather weak and we see the PSO approach suitable mostly for testing purposes, if anyone wants to try. A very powerful new method, complementary to our evolutionary algorithm, is evolutionary metadynamics [14], a hybrid of Martoňák's metadynamics and the our evolutionary approach. This method is powerful for global optimization and for harvesting low-energy metastable structures, and even for finding possible phase transition pathways. For detailed investigations of phase transition mechanisms, additional methods are implemented: variable-cell NEB method [15] and transition path method [16] in the version [17].

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 (Section 5.6.2);
  - starting structure search from known or hypothetical structures (Section 5.6.14);
  - assembling crystal structures from predefined molecules (Section 5.10).
- Efficient constraint techniques, which eliminate unphysical and redundant regions of the search space. Cell reduction technique (Oganov & Glass, 2008 [18]).
- Niching using fingerprint functions (Oganov & Valle, 2009 [19]; Lyakhov, Oganov, Valle, 2010 [20]). Section 5.6.5 for details.
- Initialization using fully random approach, or using space groups and cell splitting techniques (Lyakhov, Oganov, Valle, 2010 [20]). Use of powerful topological structure generator (Bushlanov, Blatov, Oganov, 2018 [21]).
- Prediction of the structure of nanoparticles and surface reconstructions.
- Restart functionality.

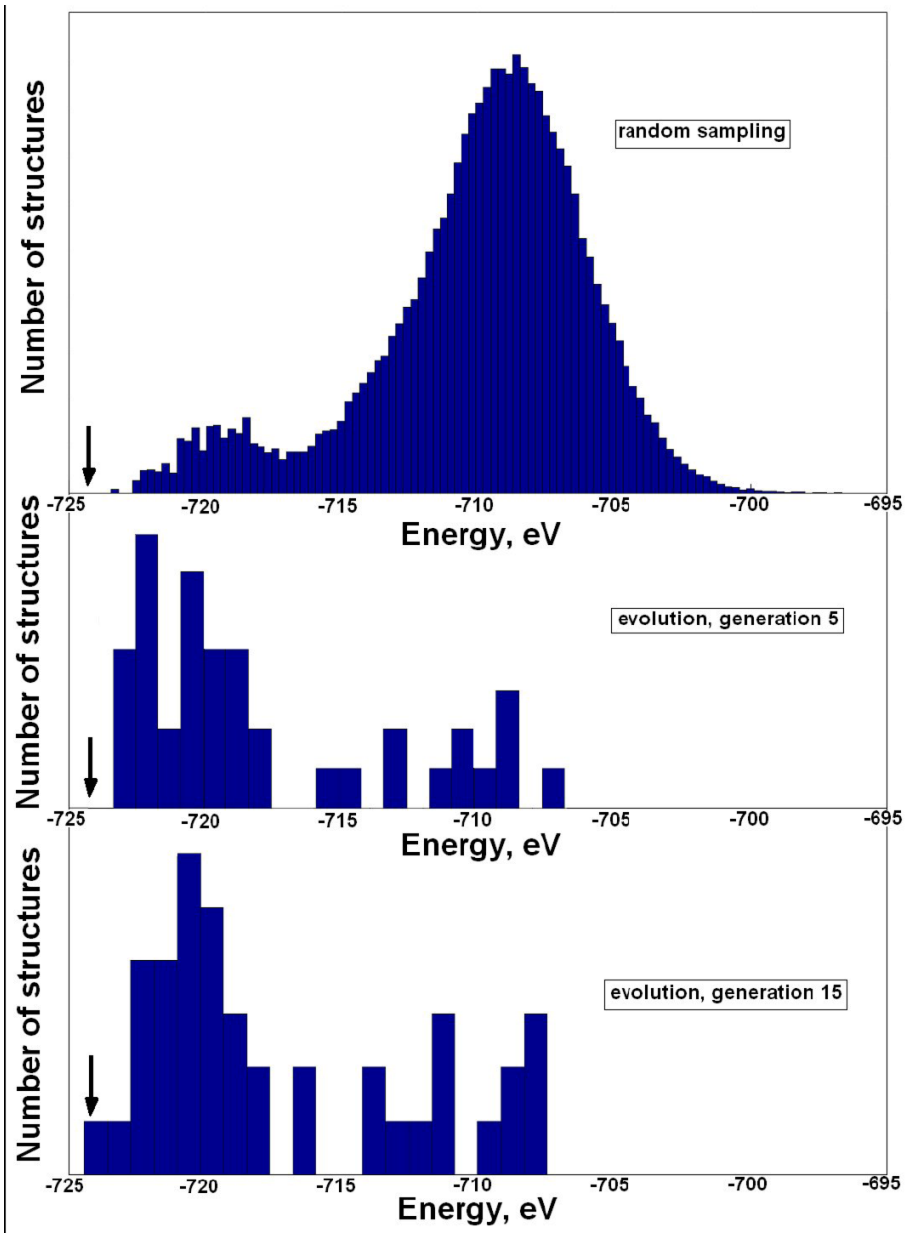

Fig. 1.1.4: Sampling of the energy surface: comparison of random sampling and USPEX for a 40-atom cell of $MgSiO_3$ with cell parameters of post-perovskite. Energies of locally optimized structures are shown. For random sampling, 1.2×10^5 structures were generated (none of which corresponded to the ground 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.

- USPEX is interfaced with VASP, GULP, LAMMPS, ABINIT, Quantum Espresso, FHIaims, MOPAC codes. See full list of supported codes in Subsection Section 2.5.
- Submission of jobs from local workstation to remote clusters and supercomputers is possible.

1.3 Key USPEX papers

1. Oganov A.R., Glass C.W. (2006). Crystal structure prediction using evolutionary algorithms: principles and applications. *J. Chem. Phys.*, **124**, 244704.
2. Oganov A.R., Stokes H., Valle M. (2011). How evolutionary crystal structure prediction works — and why. *Acc. Chem. Res.*, **44**, 227–237.
3. Lyakhov A.O., Oganov A.R., Stokes H., Zhu Q. (2013). New developments in evolutionary structure prediction algorithm USPEX. *Comp. Phys. Comm.*, **184**, 1172–1182.
4. Zhu Q., Oganov A.R., Glass C.W., Stokes H. (2012). Constrained evolutionary algorithm for structure prediction of molecular crystals: methodology and applications. *Acta Cryst. B*, **68**, 215–226.
5. Zhu Q., Li L., Oganov A.R., Allen P.B. (2013). Evolutionary method for predicting surface reconstructions with variable stoichiometry. *Phys. Rev. B*, **87**, 195317.
6. Zhu, Q., Sharma V., Oganov A.R., Ramprasad R. (2014). Predicting polymeric crystal structures by evolutionary algorithms. *J. Chem. Phys.*, **141**, 154102.
7. Lyakhov A.O., Oganov A.R., Valle M. (2010). Crystal structure prediction using evolutionary approach. In: Modern methods of crystal structure prediction (ed: A.R. Oganov), Berlin: Wiley-VCH.
8. Oganov A.R., Ma Y., Lyakhov A.O., Valle M., Gatti C. (2010). Evolutionary crystal structure prediction as a method for the discovery of minerals and materials. *Rev. Mineral. Geochem.*, **71**, 271–298.
9. Duan, D., Liu, Y., Tian, F., Li, D., Huang, X., Zhao, Z., Yu, H., Liu, B., Tian, W., Cui, T. (2014). Pressure-induced metallization of dense $(H_2S)_2H_2$ with high-T_c superconductivity. *Sci. Rep.*, **4**, 6968.
10. Bushlanov P.V., Blatov V.A., Oganov A.R. (2019). Topology-based crystal structure generator. Comp. Phys. Comm., DOI: 10.1016/j.cpc.2018.09.016.
11. Lepeshkin S.V., Baturin V.S., Uspenskii Yu.A., Oganov A.R. (2019). Method for simultaneous prediction of atomic structure of nanoclusters in a wide area of compositions. J. Phys. Chem. Lett. 10, 102-106.
12. Allahyari Z., Oganov A.R. (2018). Multi-objective optimization as a tool for materials design. In:.Handbook of Materials Modeling (ed. W. Andreoni, S. Yip). Volume 2 Applications: Current and Emerging Materials. Springer Verlag.
13. Allahyari Z., Oganov A.R. (2020). Coevolutionary search for optimal materials in the space of all possible compounds. npj Computational Materials volume 6, Article number: 55.
14. Liu X., Niu H., Oganov A.R. (2021). COPEX: co-evolutionary crystal structure prediction algorithm for complex systems. npj Comp. Mater. 7, 199

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 . 20/12/2006.

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

v.6 — Production version.

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

6.2 — Development version.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

7.4.1 — Introduced coordinate mutation based on local order [19]. 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 , . 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 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), =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), 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 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 , , 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. Released 15/10/2019.

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. Released 05/11/2020.

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. Released 08/07/2021.

2021.0 - This version is a beginning of reimplementation of previously developed techniques (with improvements, bug fixes, and more transparent code structure to facilitate further developments). In this version we have reimplemented structure/compound prediction for bulk crystals. Released 27/10/2021.

2022.0 - Continuation of USPEX pythonization effort. In this version we bring structure prediction of films, surfaces and nanoparticles. Realeased 20/08/2022.

2022.1 - Continuation of USPEX pythonization effort. New approach to calculating distance between atomic structures utilizing the same atomic fingerprints. Implemented global parent pool option. Introduced auxiliary `*.uspex` files, which allows to use seeds for molecular crystals. New optimization types. Realeased 14/11/2022.

2023.0 - Continuation of USPEX pythonization effort. Experimental active learning of interatomic potential mode. Interface for MLIP code. Realeased 23/05/2023.

Note: All regimes of 10.5 version will be reimplemented in subsequent releases.

CHAPTER

TWO

GETTING STARTED

2.1 How to obtain USPEX

USPEX can be downloaded at:

https://uspex-team.org

In the download page you will need to register and soon thereafter will receive a password for downloading USPEX.

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 [2,14,20], 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.uspex`, `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

Pythonized version of USPEX can be used on linux platform — all you need is one CPU with installed python and — using its special remote submission mechanism, USPEX will be able to connect to any remote machine and use it for calculations.

2.5 Codes that can work with USPEX

Trial structures generated by USPEX are relaxed and then evaluated by an external code interfaced with USPEX. Based on the obtained ranking of relaxed structures, USPEX generates new structures — which are again relaxed and ranked. Our philosophy is to use existing well-established *ab initio* (or classical forcefield) codes for structure relaxation and energy calculations. Currently, USPEX-2023.0 is interfaced with:

- VASP
- GULP
- LAMMPS
- Quantum Espresso
- FHI aims
- ABINIT
- MOPAC

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

2.6 How to install USPEX

USPEX v.2023.0 is compiled using Cython compiler. After you download the archive with USPEX, you need to unpack and install it using *pip* installer.

```
pip install USPEX-2023.0.1-<ver>-<ver>-linux_x86_64.whl
```

Chose one of *whl* files in archive depending on version of python you are using.

The file **random_cell** should be copied manually into any directory listed in user's *PATH* environment setting. For example into `~/.local/bin/` folder.

2.7 How to run USPEX

To run USPEX, you need to have **Python** and to have the executable of the external code on the compute nodes that you use for relaxing structures and computing their energies (see Section 2.5 for a list of supported codes). To set up your calculation, find an example (see Section 4 for a list of examples) similar to what you want to do, and start by editing `input.uspex`. The variables of this crucial file are described in Section 5 below. Then, gather the files needed for the external code performing structure relaxation in the 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.

To run USPEX just type:

```
uspex -r
```

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

CHAPTER

THREE

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 Section 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.uspex`, thoroughly described in Section 5.
- Subdirectory `~/StructurePrediction/Specific/` with VASP, GULP, *etc.* executables, and enumerated input files for structure relaxation — `INCAR_1`, `INCAR_2`, ..., and pseudopotentials. You can actually alter this filenames (see Section 5.8)
- Subdirectory `~/StructurePrediction/Seeds/` contains files with seed structures. Seed structures should be in VASP5 POSCAR format grouped into folders. Which folder will be used for which generation is specified in Section 5.6.14.
- Files with molecule definitions (see Section 5.10).
- Files with environment definitions (see Section 5.11).

3.1.1 Specific folder

Executables and enumerated input files for structure relaxation (using external codes, like VASP, GULP, ...) should be put in subdirectory

- 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. files of all relevant elements should also be in Specific folder, for instance POTCAR_O, POTCAR_C, *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.*).
- For Quantum Espresso, files qEspresso options 1, qEspresso options 1, ..., must be present. All files should be the normal QE input files with all parameters except atom coordinates, cell parameters and k-points (these will be written by USPEX at the end of the file). We recommend performing a multi-step relaxation. For instance qEspresso options 1, does a crude structure relaxation of atomic positions with fixed cell parameters, qEspresso options 2 does full structure relaxation under constant external pressure with medium precision; qEspresso options 3 and does very accurate calculations.

INCAR_* files in Specific/ folder for VASP

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

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

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

We recommend the Methfessel-Paxton smearing scheme (ISMEAR = 1). For a clearly metallic system, use ISMEAR = 1 and SIGMA = 0.1–0.2. For a clearly insulating system, we recommend ISMEAR = 1 and SIGMA starting at 0.1 (INCAR_1) and decreasing to 0.05.

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

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

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

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

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

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

3.2 Output files

These are stored in the folder `results1`, if this is a new calculation and `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 the true space group symmetry, either increase symmetry tolerances (but this can be dangerous), or re-relax your structure with increased precision.

The subdirectory contains the following files:

- `OUTPUT.txt` – summarizes input variables, structures produced by USPEX, and their characteristics.
- `parameters.uspex` — this is a copy of the file used in this calculation with some defaults explicitly writen, 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.*).
- `BESTindividuals` gives this information for the best structures from each generation.
- `convex_hull` — only for variable-composition calculations, this file gives all thermodynamically stable compositions, and their enthalpies (per atom).
- `gatheredPOSCARS` — relaxed structures (in the VASP5 POSCAR format).
- `BESTgatheredPOSCARS` — the same data for the best structure in each generation.
- `gatheredPOSCARS_unrelaxed` — gives all structures produced by USPEX before relaxation.
- `enthalpies_complete.csv` — gives the enthalpies for all structures in each stage of relaxation.
- `origin` — shows which structures originated from which parents and through which variation operators.
- `goodStructures` and `extended_convex_hull` (for fixed- and variable-composition calculations correspondingly) report all of the different structures (details) in order of decreasing stability, starting from the most stable structure and ending with the least stable.
- `goodStructures_POSCARS` and `extended_convex_hull_POSCARS` (for fixed- and variable-composition calculations correspondingly) report all of the different structures (in the VASP5 POSCAR format) in order of decreasing stability, starting from the most stable structure and ending with the least stable.

- `*.uspex` auxiliary files which complement `POSCARS` files if needed. They contain information about molecular association of atoms as well as periodical boundary conditions.
- graphical files (**.svg**) — for rapid visual assessment of the results:
  - `Energy_vs_N.svg` (`Fitness_vs_N.svg`) — energy (fitness) as a function of structure number;
  - `Energy_vs_Volume.svg` — energy as a function of volume;
  - `Variation-Operators.svg` — 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` — 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.svg`, which shows the enthalpy of formation as function of composition.

CHAPTER

FOUR

TYPICAL USE CASES

4.1 Crystal structure prediction (3D): fixed-composition

4.1.1 Highlights

This is the simplest in terms of input use case. Here we describe all its input blocks.

```
type : GlobalOptimizer
```

It tells program that we want to look for global optimum in some space. The alternative would be an optimization of some single entity, for example a phase transition pathway in the case of VCNEB calculation (it is not yet implemented in USPEX-2023.0).

```
target: {
   type : Atomistic
   conditions: {externalPressure: 100}
   compositionSpace: {
      symbols: [Mg Al O]
      blocks: [[4 8 16]]
   }
}
```

This block is specific for global optimization. Here you specify type of the space you want to investigate and its parameters. In this example type is *Atomistic*, which covers all systems composed from atoms: crystals (including molecular ones), surfaces, thin films, grain boundaries, polymers, nanoparticles. The alternative types cover chemical composition in case of Mendeleev search, transition pathway in case of phase transition global optimization, and so on. These regimes are not yet transferred to the Python version of USPEX.

The main property of the search space in any Atomistic calculation is *compositionSpace*. In this example it is a space of fixed-composition, non-molecular $Mg_4Al_8O_{16}$ structures.

The other property set for target space in the example is *conditions.externalPressure*. It has two effects: it is used in estimating the initial structure volume (useful for relaxation), and it is then used for structure relaxation. In the example conditions.externalPressure is set to 100 GPa.

```
optType: enthalpy
```

This parameter is also specific for global optimization. It tells program which property we want to optimize.

```
selection: {
   type: USPEXClassic
   popSize: 50
```

(continues on next page)

(continued from previous page)

```
   optType: (aging enthalpy)
   fractions: {
      heredity: (0.1 1.0 0.5)
      softModeMutation: (0.1 1.0 0.2)
      permutation: (0.5 1.0 0.1)
      randSym: (0.05 1.0 0.1)
      randTop: (0.05 1.0 0.1)
   }
}
```

This block, specific for global optimization, determines the optimization algorithm and its parameters. In the example the *USPEXClassic* algorithm is used, which is the evolutionary algorithm. Other algorithms: *PSO* and *Metadynamics* from USPEX-10 series are not reimplemented at the moment. Parameter *popSize* defines the number of individuals in each generation. Parameter *optType* in this section governs parents selection. It could be skipped, in which case the *optType* from *optimizer* section will be used. But often you would like to apply additional adjustments to how parents are chosen. In this case *aging* procedure (also known as antiseeds method) is applied. In this method all structures which have already been visited will be penalized by an extra energy term each time they appear again.

The *fractions* block here governs variation operators fractions in generations. Each variation operator have three parameters: minimum fraction. maximum fraction and initial weight. Program will ensure (or at least try to) that in each population the fraction of structures created with certain operator will be in range between its minimum and maximum values. The initial weights describes from which value the operator starts in the search. The weights not necessarily add up to one.

```
numGenerations: 50
stopCrit: 28
```

This two parameters are generic for all USPEX regimes. They defines total number of generation to be processed and the main halting criteria. Which is the number of generations during which optimization did not found any better solution. For example if the calculation found new global optimum on 3rd generation and then until 23rd it did not found new global optimum, it will be stopped.

```
stages: [glp glp glp glp glp]
numParallelCalcs: 10
```

```
#define glp
{type : gulp commandExecutable : 'gulp'}
```

Here you define how you do ab initio (or classical) calculations of your structures. In this example, you use GULP for relaxing structures in 5 stages. And you do 10 relaxations in parallel.

4.1.2 Example

```
{
   optimizer: {
      type : GlobalOptimizer
      target: {
         type : Atomistic
         conditions: {externalPressure: 100}
         compositionSpace: {
               symbols: [Mg Al O]
```

(continues on next page)

(continued from previous page)

```
            blocks: [[4 8 16]]
         }
      }
      optType: enthalpy
      selection: {
         type: USPEXClassic
         popSize: 50
         optType: (aging enthalpy)
         fractions: {
            heredity: (0.1 1.0 0.5)
            softModeMutation: (0.1 1.0 0.2)
            permutation: (0.5 1.0 0.1)
            randSym: (0.05 1.0 0.1)
            randTop: (0.05 1.0 0.1)
         }
      }
   }
   numGenerations: 50
   stopCrit: 28
   stages: [glp glp glp glp glp]
   numParallelCalcs: 10
}

#define glp
{type : gulp commandExecutable : 'gulp'}
```

4.1.3 Checklist

To predict crystal structure for a certain composition, you need to:

- Specify `optimizer.type: GlobalOptimizer`, `optimizer.target.type: Atomistic` and `optimizer.selection.type: USPEXClassic`.
- Specify composition in `optimizer.target.compositionSpace` block. For a fixed composition only `symbols` and `blocks` required there. See Section 5.6.1 for details.
- Optionally specify `optimizer.target.ionDistances` and `optimizer.target.conditions`. See Section 5.6 and Section 5.6.6 for details.
- Optionally specify `optimizer.target.cellUtility.cellVectors` or `optimizer.target.cellUtility.cellParameters` if you want to run calculation in fixed cell. See Section 5.6.2 for details.
- Specify feature to be optimized in `optimizer.optType` (fitness). See Section 5.5 for details.
- Additionally and optionally specify `optimizer.selection.optType`. This feature if specified will be used in selection procedure instead of `optimizer.optType`. Usually you will want to specify here the same quality as in `optimizer.optType` but with `aging` procedure involved. In case of single parameter optimization this is trivial, but if yo do pareto optimization you will have choice what features should be aged. See Section 5.5 for details.
- Specify population size in `optimizer.selection.popSize` and optionally initial population size in `optimizer.selection.initialPopSize`. See Section 5.7 for details.
- Assign fraction ranges to the variation operators to be used in the calculation in `optimizer.selection.fractions`. See Section 5.7 for details.

- Define all needed ab initio calculation sections and use them in `stages`. See Section 5.8 for details.
- Specify `numParallelCalcs`, `numGenerations` and `stopCrit` parameters. See Section 5.2 for details.

4.2 Crystal structure prediction (3D): variable-composition

4.2.1 Highlights

Most of input is the same as in previous case. Here we draw your attention to differences.

```
compositionSpace: {
   symbols: [Mg Al O]
   blocks: [[1 0 1] [0 2 3]]
   minAt: 30 maxAt 42
}
```

The first major difference is in definition of composition space. Now we have $MgO - Al_2O_3$ variable-composition system and we reflect it in the *blocks* parameter. Additionally we set up minimum and maximum number of atoms in structure.

```
optType: enthalpyCCH
```

The second major difference is the property to be optimized. Unlike in fixed-composition case we need a property which can be meaningfully compared for systems with different number of atoms. This is the *enthalpyCCH* or height above enthalpy derived composition convex hull.

```
popSize: 80
initPopSize: 200
optType: (aging enthalpyCCH)
```

The same transition form *enthalpy* to *enthalpyCCH* should be done in *selection* section. Also in this section we provide *initPopSize* besides usual *popSize*. This parameter allows to set up size of initial generation different from size of all subsequent.

4.2.2 Example

```
{
   optimizer: {
      type : GlobalOptimizer
      target: {
         type : Atomistic
         compositionSpace: {
            symbols: [Mg Al O]
            blocks: [[1 0 1] [0 2 3]]
            minAt: 30 maxAt 42
         }
      }
      optType: enthalpyCCH
      selection: {
         type: USPEXClassic
         popSize: 80
```

(continues on next page)

(continued from previous page)

```
        initPopSize: 200
        optType: (aging enthalpyCCH)
        fractions: {
           heredity: (0.1 1.0 0.5)
           softModeMutation: (0.1 1.0 0.2)
           transmutation: (0.5 1.0 0.1)
           randSym: (0.05 1.0 0.1)
           randTop: (0.05 1.0 0.1)
        }
     }
  }
  stages: [glp glp glp]
  numParallelCalcs: 10
  numGenerations: 60
  stopCrit: 20
}

#define glp
{type : gulp commandExecutable : 'gulp'}
```

4.2.3 Checklist

To predict crystal structure with for a known constituent elements, you need to:

- Specify `optimizer.type: GlobalOptimizer`, `optimizer.target.type: Atomistic` and `optimizer.selection.type: USPEXClassic`.
- Specify composition in `optimizer.target.compositionSpace` block. Besides `symbols` and `blocks` either `minAt`, `maxAt` or `range` required there. See Section 5.6.1 for details.
- Optionally specify `optimizer.target.ionDistances` and `optimizer.target.conditions`. See Section 5.6 and Section 5.6.6 for details.
- Specify feature to be optimized in `optimizer.optType` (fitness). Be sure to provide here intensive property. If you want to do stable structures prediction it should be `enthalpyCCH` (height above enthalpy derived composition convex hull) rather than `enthalpy`. See Section 5.5 for details.
- Additionally and optionally specify `optimizer.selection.optType`. This feature if specified will be used in selection procedure instead of `optimizer.optType`. Usually you will want to specify here the same quality as in `optimizer.optType` but with `aging` procedure involved. In case of single parameter optimization this is trivial, but if yo do pareto optimization you will have choice what features should be aged. See Section 5.5 for details.
- Specify population size in `optimizer.selection.popSize` and optionally initial population size in `optimizer.selection.initialPopSize`. See Section 5.7 for details.
- Assign fraction ranges to the variation operators to be used in the calculation in `optimizer.selection.fractions`. See Section 5.7 for details.
- Define all needed ab initio calculation sections and use them in `stages`. See Section 5.8 for details.
- Specify `numParallelCalcs`, `numGenerations` and `stopCrit` parameters. See Section 5.2 fro details.

4.3 Crystal structure prediction (3D): molecular crystals

4.3.1 Highlights

Here we highlight differences from previous cases.

```
compositionSpace: {symbols: [mol_h2o] blocks: [[4]]}
```

```
#define mol_h2o
{filename: 'MOL_H2O'}
```

When we do molecular crystal optimization we provide molecules rather then pure atoms as building units. To do so we define molecules in our definition sections and use introduced molecule names in *compositionSpace* instead of element symbols.

4.3.2 Example

```
{
   optimizer : {
      type : GlobalOptimizer
      target: {
         type: Atomistic
         compositionSpace: {symbols: [mol_h2o] blocks: [[4]]}
      }
      optType: enthalpy
      selection: {
         type: USPEXClassic
         popSize: 50
         optType: (aging enthalpy)
         fractions: {
            heredity: (0.1 1.0 0.5)
            softmodemutation: (0.1 1.0 0.3)
            randSym: (0.1 1.0 0.1)
            randTop: (0.1 1.0 0.1)
         }
      }
   }
   stages : [glp glp]
   numParallelCalcs : 10
   numGenerations : 50
   stopCrit: 10
}

#define glp
{
   type : gulp
   commandExecutable : gulp
   libs : ['./Specific/dreiding.lib']
}
```

(continues on next page)

(continued from previous page)

```
#define mol_h2o
{filename: 'MOL_H2O'}
```

4.3.3 Checklist

To predict crystal structure for a certain composition, you need to:

- Specify `optimizer.type: GlobalOptimizer`, `optimizer.target.type: Atomistic` and `optimizer.selection.type: USPEXClassic`.
- Define molecules in corresponding sections. Provide files describing molecules. See Section 5.10 for details.
- Specify composition in `optimizer.target.compositionSpace` block using molecule names from previous step. See Section 5.6.1 for details.
- Optionally specify `optimizer.target.ionDistances` and `optimizer.target.conditions`. See Section 5.6 and Section 5.6.6 for details.
- Optionally specify `optimizer.target.cellUtility.cellVectors` or `optimizer.target.cellUtility.cellParameters` if you want to run calculation in fixed cell. See Section 5.6.2 for details.
- Specify feature to be optimized in `optimizer.optType` (fitness). See Section 5.5 for details.
- Additionally and optionally specify `optimizer.selection.optType`. This feature if specified will be used in selection procedure instead of `optimizer.optType`. Usually you will want to specify here the same quality as in `optimizer.optType` but with `aging` procedure involved. In case of single parameter optimization this is trivial, but if yo do pareto optimization you will have choice what features should be aged. See Section 5.5 for details.
- Specify population size in `optimizer.selection.popSize` and optionally initial population size in `optimizer.selection.initialPopSize`. See Section 5.7 for details.
- Assign fraction ranges to the variation operators to be used in the calculation in `optimizer.selection.fractions`. See Section 5.7 for details.
- Define all needed ab initio calculation sections and use them in `stages`. See Section 5.8 for details.
- Specify `numParallelCalcs`, `numGenerations` and `stopCrit` parameters. See Section 5.2 fro details.

4.4 Thin films (2D)

4.4.1 Highlights

Here we highlight differences from previous cases.

```
cellUtility: {dim : 2 thickness: 5.0}
```

The code block determining dimensionality is *cellUtility*. To switch 2D regime you need to set *cellUtility.dim* to 2. The other property required for thin film prediction is the thickness of the film *cellUtility.thickness*.

```
radialDistributionUtility: {tolerance: 0.0002}
```

Our algorithm for determining individual duplicates is clibrated for 3D systems. For it to working properly for lesser dimensional systems, you need to adjust tolerance. We developing new approach to tackle this problem so users will no need to specify it in future.

```
randSymPyXtal: (0.1 1.0 0.2)
```

For lesser dimensional structures we use *randSymPyxXtal* random structure generator instead of *RandSym* and *RandTop*.

```
stages: [glp10 glp15]
```

```
#define glp10
{type : gulp commandExecutable : 'gulp' vacuumSize: 10}

#define glp15
{type : gulp commandExecutable : 'gulp' vacuumSize: 15}
```

This will tell program to use different vaccume sizes for different stages of relaxation. The vacuum layer is necessary as we emulate uperiodicity by dividing structure layers with vacuum.

4.4.2 Example

```
{
   optimizer: {
      type: GlobalOptimizer
      target: {
         type: Atomistic
         conditions: {externalPressure: 0.0}
         compositionSpace: {symbols: [Mg O] blocks: [[16 16]]}
         cellUtility: {dim : 2 thickness: 5.0}
         radialDistributionUtility: {tolerance: 0.0002}
      }
      optType: enthalpy
      selection: {
         type: USPEXClassic
         popSize: 30
         initialPopSize: 40
         optType: (aging enthalpy)
         fractions: {
            heredity: (0.3 0.7 0.5)
            softmodemutation: (0.1 0.5 0.2)
            permutation: (0.1 0.5 0.1)
            randSymPyXtal: (0.1 1.0 0.2)
         }
      }
   }
   stages: [glp10 glp15]
   numParallelCalcs: 10
   numGenerations: 30
   stopCrit: 15
}

#define glp10
{type : gulp commandExecutable : 'gulp' vacuumSize: 10}
```

(continues on next page)

(continued from previous page)

```
#define glp15
{type : gulp commandExecutable : 'gulp' vacuumSize: 15}
```

4.4.3 Checklist

To predict film (2D crystal) structure, you need to:

- Specify `optimizer.type: GlobalOptimizer`, `optimizer.target.type: Atomistic` and `optimizer.selection.type: USPEXClassic`.
- Specify composition in `optimizer.target.compositionSpace` block. See Section 5.6.1 for details.
- Optionally specify `optimizer.target.ionDistances` and `optimizer.target.conditions`. See Section 5.6 and Section 5.6.6 for details.
- Specify `cellUtility.dim: 2`.
- Specify film thickness in `cellUtility.thickness`.
- Optionally specify `optimizer.target.cellUtility.cellVectors` or `optimizer.target.cellUtility.cellParameters` if you want to run calculation in fixed cell. See Section 5.6.2 for details.
- Specify feature to be optimized in `optimizer.optType` (fitness). For variable-composition calculation provide an intensive property (`enthalpyCCH` instead of `enthalpy`). See Section 5.5 for details.
- Additionally and optionally specify `optimizer.selection.optType`. This feature if specified will be used in selection procedure instead of `optimizer.optType`. Usually you will want to specify here the same quality as in `optimizer.optType` but with `aging` procedure involved. In case of single parameter optimization this is trivial, but if yo do pareto optimization you will have choice what features should be aged. See Section 5.5 for details.
- Specify population size in `optimizer.selection.popSize` and optionally initial population size in `optimizer.selection.initialPopSize`. See Section 5.7 for details.
- Assign fraction ranges to the variation operators to be used in the calculation in `optimizer.selection.fractions`. See Section 5.7 for details.
- Define all needed ab initio calculation sections and use them in `stages`. Do not forget to specify `vacuumSize` which should be sufficiently large to prevent interaction between periodic images of slabs. See Section 5.8 for details.
- Specify `numParallelCalcs`, `numGenerations` and `stopCrit` parameters. See Section 5.2 fro details.

4.5 Surface reconstructions (2D): fixed-composition

Variable-composition calculation is possible but input is messy right now. It will be documented in the nex release.

4.5.1 Highlights

Here we highlight differences from previous cases.

```
cellUtility: {
   dim : 2
   thickness: 5.0
   supercellDegree: 4
}
```

Here we tell program that we want 2D cells constructed from 4 tiles.

```
environmentUtility: {environments: [substrate]}
```

```
#define substrate
{
   type: substrate
   file: './POSCAR_SUBSTRATE'
   pbc: (1 1 0)
   bufferThickness: 3.0
}
```

Here we set up substrate. The structure is read from file './POSCAR_SUBSTRATE'. It has buffer layer of size 3.0Å.

4.5.2 Example

```
{
   optimizer: {
      type: GlobalOptimizer
      target: {
         type: Atomistic
         conditions: {externalPressure: 0.0}
         compositionSpace: {symbols: [Mg O] blocks: [[16 16]]}
         cellUtility: {
            dim : 2
            thickness: 5.0
            supercellDegree: 4
         }
         radialDistributionUtility: {tolerance: 0.0002}
         environmentUtility: {environments: [substrate]}
      }
      optType: enthalpy
      selection: {
         type: USPEXClassic
         popSize: 30
         initialPopSize: 40
         optType: (aging enthalpy)
         fractions: {
            heredity: (0.3 0.7 0.5)
            softmodemutation: (0.1 0.5 0.2)
            permutation: (0.1 0.5 0.1)
            randSymPyXtal: (0.1 1.0 0.2)
```

(continues on next page)

(continued from previous page)

```
        }
      }
   }
   stages: [glp10 glp15]
   numParallelCalcs: 10
   numGenerations: 30
   stopCrit: 15
}


 #define glp10
 {type : gulp commandExecutable : 'gulp' vacuumSize: 10}

 #define glp15
 {type : gulp commandExecutable : 'gulp' vacuumSize: 15}

 #define substrate
 {
    type: substrate
    file: './POSCAR_SUBSTRATE'
    pbc: (1 1 0)
    bufferThickness: 3.0
 }
```

4.5.3 Checklist

To predict surface reconstructions, you need to:

- Specify `optimizer.type: GlobalOptimizer`, `optimizer.target.type: Atomistic` and `optimizer.selection.type: USPEXClassic`.
- Specify composition in `optimizer.target.compositionSpace` block. See Section 5.6.1 for details.
- Optionally specify `optimizer.target.ionDistances` and `optimizer.target.conditions`. See Section 5.6 and Section 5.6.6 for details.
- Specify `cellUtility.dim: 2`.
- Specify surface region thickness in `cellUtility.thickness`.
- Specify feature to be optimized in `optimizer.optType` (fitness). For variable-composition calculation provide intensive property (`enthalpyCCH` instead of `enthalpy`). See Section 5.5 for details.
- Additionally and optionally specify `optimizer.selection.optType`. This feature if specified will be used in selection procedure instead of `optimizer.optType`. Usually you will want to specify here the same quality as in `optimizer.optType` but with `aging` procedure involved. In case of single parameter optimization this is trivial, but if yo do pareto optimization you will have choice what features should be aged. See Section 5.5 for details.
- Specify population size in `optimizer.selection.popSize` and optionally initial population size in `optimizer.selection.initialPopSize`. See Section 5.7 for details.
- Assign fraction ranges to the variation operators to be used in the calculation in `optimizer.selection.fractions`. See Section 5.7 for details.

- Define all needed ab initio calculation sections and use them in `stages`. Do not forget to specify `vacuumSize` which should be sufficiently large to prevent interaction between periodic images of slabs. See Section 5.8 for details.
- Define substrate in corresponding section and use it in `optimizer.target.environmentUtility.environments`. Provide a file containing substrate in VASP5 POSCAR format, specify `pbc` and `bufferThickness`. The file can be constructed using online utility: Substrate.
- Specify `numParallelCalcs`, `numGenerations` and `stopCrit` parameters. See Section 5.2 fro details.

4.6 Nanoparticles prediction (0D)

4.6.1 Highlights

Here we highlight differences from previous cases.

```
cellUtility: {dim : 0}
```

The dimension of the problem here is 0.

4.6.2 Example

```
{
   optimizer: {
      type: GlobalOptimizer
      target: {
         type: Atomistic
         conditions: {externalPressure: 0.0}
         compositionSpace: {symbols: [Mo] blocks: [[36]]}
         cellUtility: {dim : 0}
      }
      optType: enthalpy
      selection: {
         type: USPEXClassic
         popSize: 40
         optType: (aging enthalpy)
         fractions: {
            heredity: (0.3 0.7 0.5)
            softmodemutation: (0.1 0.5 0.3)
            randSymPyXtal: (0.1 1.0 0.2)
         }
      }
   }
   stages: [glp10 glp11 glp12]
   numParallelCalcs: 20
   numGenerations: 60
   stopCrit: 20
}

#define glp10
{type : gulp commandExecutable : 'gulp' vacuumSize: 10}
```

(continues on next page)

(continued from previous page)

```
#define glp11
{type : gulp commandExecutable : 'gulp' vacuumSize: 11}

#define glp12
{type : gulp commandExecutable : 'gulp' vacuumSize: 12}
```

4.6.3 Checklist

To predict structure of nanoparticles (clusters), you need to:

- Specify `optimizer.type: GlobalOptimizer`, `optimizer.target.type: Atomistic` and `optimizer.selection.type: USPEXClassic`.
- Specify composition in `optimizer.target.compositionSpace` block. See Section 5.6.1 for details.
- Optionally specify `optimizer.target.ionDistances`. See Section 5.6 for details.
- Specify `cellUtility.dim: 0`.
- Specify feature to be optimized in `optimizer.optType` (fitness). See Section 5.5 for details.
- Additionally and optionally specify `optimizer.selection.optType`. This feature if specified will be used in selection procedure instead of `optimizer.optType`. Usually you will want to specify here the same quality as in `optimizer.optType` but with `aging` procedure involved. In case of single parameter optimization this is trivial, but if yo do pareto optimization you will have choice what features should be aged. See Section 5.5 for details.
- Specify population size in `optimizer.selection.popSize` and optionally initial population size in `optimizer.selection.initialPopSize`. See Section 5.7 for details.
- Assign fraction ranges to the variation operators to be used in the calculation in `optimizer.selection.fractions`. See Section 5.7 for details.
- Define all needed ab initio calculation sections and use them in `stages`. Do not forget to specify `vacuumSize` which should be sufficiently large to prevent interaction between periodic images of slabs. See Section 5.8 for details.
- Specify `numParallelCalcs`, `numGenerations` and `stopCrit` parameters. See Section 5.2 fro details.

4.7 Machine learning interatomic potentials.

4.7.1 Explanation

In this Example, we first produce the first set of 1000 structures using random topological structure generator (which is particularly suited for producing highly ordered and diverse structures). For these unrelaxed structures we do a DFT calculation of the energy, forces and stresses. These are then used for training an initial ML potential. After that, the next 100 random topological structures are produced. Only if a structure is found to be sufficiently dissimilar to previously produced configurations, it is included in a training dataset and its energy, forces and stresses are computed at DFT level. If a structure is similar to the previously collected configurations, then it is relaxed and (if it is still similar to the previously collected configurations) a molecular dynamics calculation is run, for example, at 1500 K. If any configuration is encountered that differs from the training dataset, then a DFT calculation is performed and ML potential is retrained. This process stops when the ML remains unchanged (i.e. when no new configurations are encountered) for sufficiently many generations (=stopCrit generations).

4.7.2 Highlights

Here we describe new blocks.

```
type : ModelOptimizer
```

It tells program that we want to train a machine learning model on some space.

```
popSize: 100
initialPopSize: 1000
fractions: {
            randTop: 1.0
           }
```

Parameter *popSize* defines the number of individuals in each generation except initial. Parameter *initialPopSize* defines the number of individuals in initial generation. The *fractions* block here governs generators fractions in generations. In our example we generate all individuals with topological random structure generator.

```
model: {
    type: External
    interface: mlip_trainer
}
```

Here we tell program that we want to train a model implemented in external program which we want to communicate to via an interface `mlip_trainer` defined in corresponding section.

```
#define sample_forces
{
    stageType: populationProcessor
    tag: forces
    stages: [vasp_forces]
    inputKey: trajectory
    numParallelCalcs: 10
}
```

Here we define a stage in our calculation which takes attribute `trajectory` from an individual and treats it as a nested population applying stage `vasp_forces` to each element there.

```
#define mlip_trainer
{
    type: mlip
    mode: train
    tag: train
    potential: './Training/La_H_24g_vasp.mtp'
    trainingSet: './Training/set.cfg'
    commandExecutable: 'mpirun mlp'
    specorder: [La H]
}
```

Here we define a stage which calls external program `mlp` to train a potential. This definition is used in `ModelOptimizer` block of main section.

4.7.3 Example

```
{
    optimizer: {
        type: ModelOptimizer
        target: {
            type: Atomistic
            compositionSpace: {symbols: [La H]
                               blocks: [[1 0] [0 1]]
                               minAt: 2 maxAt: 22}
            conditions: {externalPressure : 170}
            ionDistances : {'La H': 1.905
                            'La La': 3.07
                            'H H': 0.74}
        }
        popSize: 100
        initialPopSize: 1000
        fractions: {
            randTop: 1.0
        }
        model: {
            type: External
            interface: mlip_trainer
        }
    }
    stages: [md selection sample_forces]
    numParallelCalcs: 20
    numGenerations: 500
    stopCrit: 25
}

#define md
{
    type : lammps
    mlip: './Training/La_H_24g_vasp.mtp'
    commandExecutable: 'mpirun lmp_mpi -in lammps.in; cat sampled.cfg_* > sampled.cfg'
    targetProperties: [trajectory]
    specorder: [La H]
}

#define selection
{
    type: mlip
    mode: select_add
    potential: './Training/La_H_24g_vasp.mtp'
    trainingSet: './Training/set.cfg'
    commandExecutable: 'mpirun mlp'
    specorder: [La H]
}

#define sample_forces
{
    stageType: populationProcessor
```

(continues on next page)

(continued from previous page)

```
    tag: forces
    stages: [vasp_forces]
    inputKey: trajectory
    numParallelCalcs: 10
}

#define vasp_forces
{
    stageType: execute
    type: vasp
    tag: forces
    kresol: 0.03
    commandExecutable: 'mpirun vasp_gpu'
    targetProperties: [structure energy forces]
}

#define mlip_trainer
{
    type: mlip
    mode: train
    tag: train
    potential: './Training/La_H_24g_vasp.mtp'
    trainingSet: './Training/set.cfg'
    commandExecutable: 'mpirun mlp'
    specorder: [La H]
}
```

CHAPTER

FIVE

INPUT OPTIONS. THE `INPUT.USPEX` FILE

The `input.uspex` file has json-like syntax and hierarchical structure representing modular nature of USPEX program. 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 should always be specified. Please consult online utilities at https://uspex-team.org/online_utilities/ — these help to prepare input and analyze some of results. Section 6 of this Manual briefly discusses these utilities.

In order to make structure of parameters and blocks of the `input.uspex file` more visual, we have prepared an example code template, which include every block and parameter. You can see it in Section 7.1 .

5.1 The `input.uspex` file syntax

The `input.uspex` consists of main section and a number of definition sections. Main section is mandatory and precedes definition sections. Definition sections are optional. There could be any number of definition sections. Each definition section starts with define name line.

```
{...}

#define name1
{...}

#define name2
{...}

#define name3
{...}
...
```

Main input section is described in Section 5.2, Section 5.3, Section 5.5, Section 5.6 and Section 5.7.

Definition sections correspond to parameters of ab initio calculations (Section 5.8), submission parameters (Section 5.9), molecule definitions (Section 5.10) and environment definitions (Section 5.11).

Each section is a dictionary of key, value pairs.

```
{
    key1 : value1
    key2 : value2
    key3 : value3
    ...
}
```

Such pairs correspond to different parameters or blocks of parameters. Input parameters are grouped into blocks recursively. Each block reflects some aspect of calculation. So it should be quite intuitive to prepare such input file.

Parameters can have numeric, string, sequence or map values.

```
numGenerations : 50,
stages: [vasp1 vasp2 vasp3]
ionDistances: {'C C': 2.0 'C H': 1.2 'H H': 0.7}
```

Blocks of parameters look as follows:

```
compositionSpace: {
    symbols: ...
    blocks: ...
}
```

There are two types of sequences: lists `[]` and tuples `()`. Tuples are expected to have fixed length when lists can have arbitrary one.

5.2 General calculation parameters

The most general algorithm used in all USPEX modes is iteration. At each iteration step program deals with a number of systems also called individuals. And the set of these systems is called population or generation. Such iteration is controlled by two parameters: `numGenerations` and `stopCrit`.

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

Default:

Format:

```
numGenerations : 50
```

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

Default:

Format:

```
stopCrit : 20
```

USPEX job with population consists of two parts: creation and relaxation.

Creation is controlled by block `optimizer`. This block depends on the type of optimization. For global optimum search it is `GlobalOptimizer`. In later releases `VCNEB` optimizer type will be available for optimization of a single phase transition pathway.

`optimizer :` Specifies the type of calculation and its parameters

Available types:

- `GlobalOptimizer` | global search algorithm see Section 5.3.
- `ModelOptimizer` | model training algorithm see Section 5.4.

Default:

Format:

```
optimizer : {type: GlobalOptimizer ...}
```

For population relaxation 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).

`stages :` List of relaxation stages to be applied to each individual in population. The names of definition sections should be used in this sequence. See Section 5.8 for information on writing this section.

Default:

Format:

```
stages : [glp glp glp]
```

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

Default:

Format:

```
numParallelCalcs : 10
```

One may specify output file according to their convenience.

`output :` Specifies the content and title of the columns in the output. These columns will be used whenever details of an individual will be printed (`Individuals`, `goodStructures`, etc.).

Default: depends on calculation type

Format:

```
output: { columns: [   (simpleMoleculeUtility.composition 'Composition')
              (radialDistributionUtility.structureOrder 'Structure order')
              (radialDistributionUtility.averageOrder 'Average order')
              (radialDistributionUtility.quasientropy 'Quasientropy')]
          }
```

---

outputRefreshDelay : Time (in seconds) between file updates in the results folder

Default: 120

Format:

```
outputRefreshDelay : 30
```

5.3 Global optimization search

Global optimization is one of possible types of calculation in USPEX (see `optimizer` in Section 5.2). As the title suggests it performs search for global optimum of some property over certain configuration space. Such search is done by sampling this space iteration by iteration. Sample of this space is called population. Target configuration space is described with parameters block `target`, property to be optimized with `optType` parameter and sampling method with `selection` block.

```
{
    type: GlobalOptimizer
    target: {...}
    selection: {...}
    optType: ...
}
```

---

target : Specifies the target space of search

Default:

Available types:

- `Atomistic` – structure prediction (see Section 5.6). This target covers all regimes which deals with structures consisting of atoms including molecular crystals, various dimensions and environments (like substrate). Use case examples: crystals (atomic and molecular), thin films, surfaces, nanoparticles etc.

Format:

```
target : {type: Atomistic ...}
```

---

selection : Specifies the evolutionary algorithm to be used

Default:

Available algorithms:

- `USPEXClassic` – full evolutionary algorithm (see Section 5.7).

Format:

```
selection : {type: USPEXClassic ...}
```

---

optType: This parameter specifies the property (or properties) that you want to optimize (see Section 5.5)

Default:

Format:

```
optType : enthalpy
```

5.4 Model training

In model training mode USPEX samples target space (e.g., crystal structures of certain compositional range) and retrains machine learning model (e.g., of interatomic potential) with training set generation by generation. The process of training stops when the algorithm becomes unable to produce individuals sufficiently different from those already used in training the model.

```
{
    type: ModelOptimizer
    target: {...}
    popSize: ...
    initialPopSize: ...
    fractions: {...}
    model: {...}
}
```

---

target : Specifies the target space of search

Default:

Available types:

- `Atomistic` – same target space as in structure prediction (see Section 5.6). This target deals with structures consisting of atoms including molecular crystals, various dimensions and environments (like substrate).

Format:

```
target : {type: Atomistic ...}
```

---

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

***Format*:**

```
popSize: 100
```

initialPopSize The number of structures in the initial generation.

Default: equal to `popSize`

***Format*:**

```
initialPopSize: 1000
```

fractions This parameter defines percentages of generators.

***Format*:**

```
fractions : {
            randTop: 0.5
            randSym: 0.3
            randSymPyxtal: 0.2
            }
```

---

model Specifies model to be trained.

Available types:

- `External` – the model is implemented in stand alone program. An `interface` is used to communicate with it. The names of definition section should be used in `interface` field. See Section 5.8 for information on writing this section.

***Format*:**

```
model : {
            type: External
            interface: ...
        }
```

5.5 Optimization type

Optimization type specification is designed as powerful method of writing functions inside input file. Such functions looks like

```
(func arg1 arg2 ...)
```

Here `arg1`, `arg2` could be functions themselves. So this procedure could be recursive. For example:

```
(pareto (aging enthalpy) (negate radialDistributionUtility.structureOrder))
```

This means that parameter to be optimized will be calculated for each structure by following procedure. Enthalpy will be taken from structutre directly, parameter `structureOrder` will be calculated using `radialDistributionUtility` module. Then `aging` function will be applied to enthalpy and `negate` function to `structureOrder`. Then finally optimization parameter will be determined via `pareto` function with two arguments: aged enthalpy and negated structure order.

For now there are following functions available:

Function	Description
`negate`	takes opposite value
`aging`	apply penalties to old structures (not tunable at current release)
`pareto`	calculate Pareto front in space of given arguments.

Available base parameters:

Name	Description
enthalpy	enthalpy of system
enthalpyCCH	enthalpy per block above convex hull (for variable composition)
enthalpyCS	enthalpy per block above best for the same composition
simpleMoleculeUtility.density	structre density
cellUtility.volume	volume of system unit cell (for 3D)
cellUtility.area	area of system unit cell (for 2D)
cellUtility.length	Period of system unit cell (for 1D)
bondUtility.hardness	Structure hardness
radialDistributionUtility.structureOrder	degree of order
radialDistributionUtility.quasientropy	structural quasientropy
elasticML.bulkModulus	Elastic bulk modulus calculated using ML model.
elasticML.shearModulus	Elastic shear modulus calculated using ML model.
elasticML.youngsModulus	Elastic Young's modulus calculated using ML model.
elasticML.poissonsRatio	Elastic Poisson's ratio calculated using ML model.
elasticML.pughsRatio	Elastic Pugh's modulus ratio calculated using ML model.
elasticML.vickersHardness	Elastic Vickers hardness calculated using ML model.
elasticML.fractureToughness	Elastic fracture toughness calculated using ML model.

Providing only base parameter as optType is also a valid option.

5.6 Crystal structure prediction

When target type in Section 5.3 is set to Atomistic USPEX performs global optimization in the space of structures consisting of atoms, Historically the first property which USPEX optimized was enthalpy and search was for stable crystal structures. Now this target covers atomic and molecular crystals in various dimensions and environments.

Typical target block in this mode looks like

```
{
    type: Atomistic
    compositionSpace: {symbols : [Mg Al O] blocks: [[4 8 16]]}
    conditions: {externalPressure : 100}
}
```

Besides obligatory compositionSpace block there are number of optional parameter blocks.

---

compositionSpace Describes the identity of each type of atom or molecule and specifies the number of species of each type. For details see Section 5.6.1.

cellUtility Properties of the unit cell. For details see Section 5.6.2.

bondUtility Constants and constraints related to interatomic distances. For details see Section 5.6.3.

environmentUtility Set up environments (such as substrates) used in the calculation. For details see Section 5.6.4.

radialDistributionUtility Properties for radial distribution fingerprint calculation. For details see Section 5.6.5.

conditions Properties of environment in general (currently only external pressure). For details see Section 5.6.6

heredity Properties of heredity variation operator. For details see Section 5.6.7

randSym Properties of symmetrical random structure generator. For details see Section 5.6.8

randSymPyXtal Properties of PyXtal random structure generator. For details see Section 5.6.9

randTop Properties of topological random structure generator. For details see Section 5.6.10

permutation Properties of permutation variation operator. For details see Section 5.6.11

transmutation Properties of transmutation variation operator. For details see Section 5.6.12

softmodemutation Properties of softmutation variation operator. For details see Section 5.6.13

seeds Properties of seeds for the calculation. For details see Section 5.6.14

5.6.1 Composition space

Examples:

Fixed-compositioin prediction for $Mg_4Al_8O_{16}$.

```
{
    symbols : [Mg Al O]
    blocks: [[4 8 16]]
}
```

Fixed-compositioin prediction with two molecules of type `mol_1` and two of type `mol_2` in unit cell.

```
{
    symbols : [mol_1 mol_2]
    blocks: [[2 2]]
}
```

Variable-composition prediction for the $MgO - Al_2O_3$ system, where the minimum number of atoms is 8 and the maximum 20.

```
{
    symbols : [Mg Al O]
    blocks: [[1 0 1] [0 2 3]]
    minAt : 8
    maxAt : 20
}
```

Variable-composition prediction for the $MgO - Al_2O_3$ system, where the minimum number of atoms is 8 and the maximum 20. With additional constraint that block Al_2O_3 should be present in system minimum 1 and maximum 3 times.

```
{
    symbols : [Mg Al O]
    blocks: [[1 0 1] [0 2 3]]
    range: [(0 10) (1 3)]
    minAt : 2
    maxAt : 20
}
```

Variable-composition prediction for the $MgO - Al_2O_3$ system. Block MgO should be present in system minimum 2 and maximum 5 times. Block Al_2O_3 should be present in system minimum 1 and maximum 3 times.

```
{
    symbols : [Mg Al O]
    blocks: [[1 0 1] [0 2 3]]
    range: [(2 5) (1 3)]
}
```

`symbols` Describes the identity of each type of atom or molecule. The value is a list of chemical elements or molecule names.

Default: no default

***Format*:**

```
symbols : [mol_1 mol_2 O]
```

Molecule names should be defined in molecule definition sections Section 5.10.

`blocks` Specify compositional building blocks.

Default: no default

***Format*:**

```
blocks : [[1 0 1] [0 2 3]]
```

`range` Specify ranges for each block.

Default: If `maxAt` is set then each block can be taken from 0 to `maxAt` devided by size of block, otherwise each block is taken one time.

***Format*:**

```
range : [(1 10) (1 10)]
```

`maxAt` Maximal number of atoms.

Default: If `range` is set, then maximum of upper bound in `range` for each block times size of block.

***Format*:**

```
maxAt : 40
```

minAt Minimal number of atoms.

Default: If `range` is set, then sum of lowest bound in `range` for each block times size of block.

***Format*:**

```
minAt : 2
```

5.6.2 Unit cell properties

cellVolume Initial volume of the unit cell.

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

***Format*:**

```
cellVolume : 125.0
```

Note: 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.

Note: You can also use online program https://uspex-team.org/online_utilities/volume_estimation. Users can also input the volumes manually.

Note: 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.

cellVectors Unit cell vectors.

Note: These are used in fixed-cell calculations.

Note: You should provide number of cell vectors equal to dimensionality of the problem.

Default: by default cell vectors are variable.

***Format*:**

```
cellVectors: [[1.0, 0.0, 0.0], [0.0, 1.0, 0.0], [0.0, 0.0, 1.0]]
```

cellParameters Unit cell vectors.

Note: This parameter is used in fixed-cell calculations.

Default: by default cell vectors are variable.

***Format*:**

```
cellParameters: {a: 2.474, b: 8.121, c: 6.138, alpha: 90.0, beta: 90.0,␣
↪gamma: 90.0}
```

dim

Specifies unit cell dimension.

Default: 3

***Format*:**

```
dim : 3
```

thickness

Thickness of the surface region or film (for 2D), or maximum linear size of nanoparticle (0D)

Default: 2.0 Å

***Format*:**

```
thickness: 3.5
```

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

Default: 1

***Format*:**

```
supercellDegree: 2
```

axis Vector normal to the plane of the 2D structure.

Default:

***Format*:**

```
axis : (0.0 0.0 1.0)
```

5.6.3 Constants and constraints related to interatomic distances.

ionDistances: 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 1.2Å, and $0.45 \times (V_A^{1/3} + V_B^{1/3})$ in molecular calculations, where V_A and V_B are default volumes of atom *A* and *B* estimated in USPEX.

***Format*:**

```
ionDistances : {'Mg Mg': 1.0 'Mg Si': 1.0 'Mg O': 0.8 'Si Si': 1.0 'Si O
↪': 0.8 'O O': 1.0}
```

Note: If the compound in the example above is $MgSiO_3$, the matrix reads as follows: the minimum Mg—Mg, Mg—Si, Si—Si and O—O distances allowed in a newly generated structure are 1.0 Å while the minimum Mg—O and Si—O distances are 0.7 Å . You can use this keymatrix to incorporate further system-specific information: *e.g.*, if you know that C 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. Commonly used computational methods (pseudopotentials, PAW, LAPW, and many parametric forcefields) fail when the interatomic distances are too small.

5.6.4 Environment utility

environments List of environments. Names used here should be defined in the corresponding definition sections, as described in Section 5.11. Every individual created through random structure generation gets an environment picked up from this list. Individuals created through other variation operators inherit their environments from parents.

***Format*:**

```
environments:  [substrate]
```

5.6.5 Radial distribution based fingerprint

For details on fingerprint functions we refer reader to Ref [19].

Rmax Distance cuttoff (in Ångstroms).

Default: 10.0

***Format*:**

```
Rmax : 10
```

delta Discretization (in Ångstoms) of the fingerprint.

Default: 0.08

***Format*:**

```
delta : 0.08
```

sigma Gaussian broadening of interatomic distances (in Ångstoms).

Default: 0.03

***Format*:**

```
sigma : 0.03
```

tolerance Specifies the minimum 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).

Default: 0.008

***Format*:**

```
tolerance : 0.2
```

legacy If true switch to the old style cosine distance between structures.

Default: False

***Format*:**

```
legacy : True
```

5.6.6 Conditions

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

Default: 0.0

***Format*:**

```
externalPressure: 0.00001
```

Note: **Please:** do not specify it in relaxation files in the `Specific` folder.

5.6.7 Heredity

nslabs Heredity operator might be done traditionally, when both parent structures cut into two pieces. or in 'zebra' way. In this case nslabs determines number of slabs into which parents are cut.

Default: 2 for fixed composition. For variable composition we take length of structure in the direction orthogonal to the cut and divide it by average diameter of atom (double covalent radius).

Format:

```
nslabs: 5
```

5.6.8 Symmetrical random generator

nsym Possible symmetry groups for symmetric random structure generator for crystals (space groups), layer groups for 2D-crystals, wallpaper groups for surfaces, or point groups for clusters. A certain number of structures will be produced using randomly selected groups from this list, using randomly generated lattice parameters and atomic coordinates. During this process special Wyckoff sites can be produced from general positions (Fig. 5.6.1, see Ref. [14] for details)

Default:

- For 3D crystals: 2-230

Format:

```
nsym: '16-74'
```

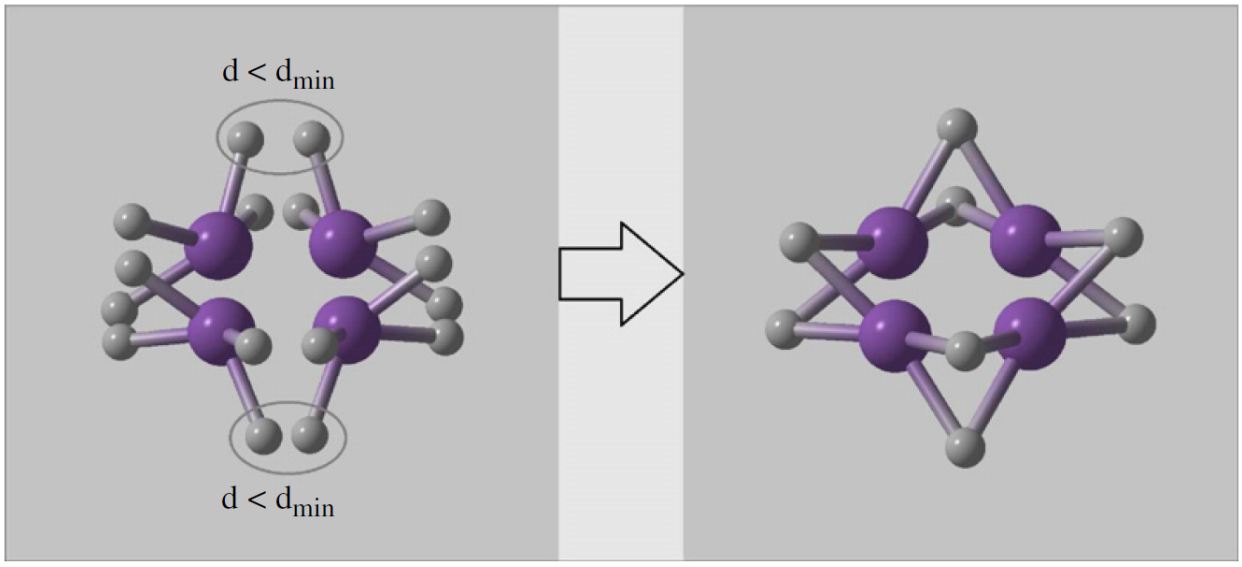

Fig. 5.6.1: **Example of random symmetric structure generation and merging atoms onto special Wyckoff positions**.

splitInto 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: [2, 4]
```

Subcells introduce extra translational (pseudo)symmetry. In addition to this, each subcell can be built using symmetric random structure generator developed by A.R. Oganov and H.T. Stokes and implemented by H.T. Stokes (see Reference [14]).

5.6.9 RandSymPyXtal

nsym Possible symmetry groups for symmetric random structure generator for crystals (space groups), layer groups for 2D-crystals, wallpaper groups for surfaces, or point groups for clusters. A certain number of structures will be produced using randomly selected groups from this list, using randomly generated lattice parameters and atomic coordinates. During this process special Wyckoff sites can be produced from general positions (Fig. 5.6.1)

***Default*:**

- For 3D crystals: 2-230

***Format*:**

```
nsym: '16-74'
```

5.6.10 Topological random generator

maxSupersize When trying to generate structure with certain number of atoms topological random generator takes from database of topologies all entries number of nodes in which fits the required one up to the factor of `maxSupersize`.

***Default*:**

- 4

***Format*:**

```
maxSupersize: 9
```

---

supercells List of allowed supercells. When trying to generate structure with certain number of atoms topological random generator takes from database of topologies all entries which when replicated according `supercells` give required number of atoms.

***Default*:**

- all supercells are allowed, up to `maxSupersize`.

***Format*:**

```
supercells: [(2 2 1) (1 2 2) (2 1 2)]
```

5.6.11 Permutation

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

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

Format:

```
howManySwaps: 5
```

specificSwaps Specifies which atom types you allow to swap in permutation.

Default: No specific swaps and all atoms are permutable

Format:

```
specificSwaps: [1 2]
```

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.

5.6.12 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.

howManyTrans 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 1 to the fractional parameter howManyTrans.

Default: 0.2

Format:

```
howManyTrans: 0.2
```

5.6.13 Softmutation

degree The maximum displacement in softmutation in angstoms. The displacement vectors for softmutation are scaled so that the largest displacement magnitude equals degree.

Default: 3×(average atomic radius)

***Format*:**

```
degree: 0.1
```

5.6.14 Seeds

This feature requires aditional input files to be provided.

generations List of generations into which seeds will be injected during the search.

Default: No default

***Format*:**

```
generations : [0 5 10]
```

---

seedsFolders List of paths to folders which contain seeds files to be injected during the search. In the same order as generations above.

Default:

***Format*:**

```
seedsFolders : ['./Seeds/1' './Seeds/2' './Seeds/3']
```

5.7 Evolutionary algorithm USPEX

When doing global optimization search (see Section 5.3) one can choose between several algorithms. In current release only evolutionary algorithm is implemented – which is the most efficient and reliable one. Particle swarm optimization and evolutionary metadynamics will be made available soon.

globalParentsPool When true algorithm will choose parents from pool of all good structures. When false – just from the previous generation.

Default: False for fixed composition. True for variable composition.

***Format*:**

```
globalParentsPool: True
```

---

popSize 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 for variable composition). The upper limit is 60. Usually, you can trust these default settings. popSize: 20

***Format*:**

```
popSize: 20
```

initialPopSize The number of structures in the initial generation.

Default: equal to populationSize

***Format*:**

```
initialPopSize: 20
```

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.

---

bestFrac Fraction of the current generation that shall be used as potential parents to produce the next generation.

Default: 0.7

***Format*:**

```
bestFrac: 0.7
```

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

---

howManyDiverse Defines how many good and diverse structures will survive into the next generation.

Default: 0.15× popSize

***Format*:**

```
howManyDiverse: 3
```

We perform clustering of the population into specified number of groups here. And take the fittest representative from each group

---

optType This keyblock specifies the property that you wish to use for selecting potential parents for new generation. See Section 5.5.

Default: same as in global optimizer section.

***Format*:**

```
optType : (aging enthalpy)
```

Fittness can differ from the property to be optimized, if one uses aging procedure.

---

fractions This parameter defines allowed percentages of each variation operator. The first value is minimum percentage of structures in each generation produced by this operator. The second — maximum percentage. The third value is weight (not the percentage) of the operator in the 1st generation.

***Format*:**

```
fractions : {
            heredity: (0.3 0.7 0.5)
            softmodemutation: (0.1 0.3 0.2)
            randTop: (0.1 0.5 0.3)
            }
```

The fractions of operators evolve during the calculation, so that the more successful operators gain weight at the expense of the less successful operators, but within the limits specified here.

5.8 Details of *ab initio* calculations sections

Typical ab initio definition section looks like

```
#define vasp1
{
type: vasp
commandExecutable: 'vasp'
kresol : 0.12
vacuumSize: 10    #only for 0- 1- or 2- dimensional calculations
}
```

It contains type of interface for external program package, common parameters (like `commandExecutable`, see Section 5.8.1), package specific parameters (like `kresol`) and `vacuumSize` which should be specified only for low-dimensional systems and should be not specified for 3D-crystals.

The definiton name (`vasp1` in the example above) should be used in *stages* parameter of general parameters Section 5.2.

Available types of interfaces are:

- `vasp` — VASP interface, see Section 5.8.2.
- `gulp` — GULP interface, see Section 5.8.3.
- `lammps` — LAMMPS interface, see Section 5.8.4.
- `qe` — Quantum Espresso interface, see Section 5.8.5.
- `abinit` — Abinit interface, see Section 5.8.6.
- `aims` — FHIaims interface, see Section 5.8.7.
- `mopac` — mopac interface, see Section 5.8.8.
- `mlip` — mlip interface, see Section 5.8.9.

5.8.1 Common interface parameters.

commandExecutable Specifies executable for a given code

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

***Format*:**

```
commandExecutable : 'mpirun vasp'
```

---

taskManager Specifies name of task manager to be used for submission.

Default: By default no task manager is used.

***Format*:**

```
taskManager : TM
```

Note: Usually you define one task manager and use it in several stages, but it can be useful to use different task managers for different stages. For details see Section 5.9

---

tag Optional identifier for stage.

Default: equal to index number of stage.

***Format*:**

```
tag : final
```

Note: This is useful if you want to highlight or distinguish a stage. This tag will appear as suffix for a job in job submission system and in calculation folder created for this job. If this parameter is set then files in `Specific/` folder should use it in their suffixes instead of index number of stage. For eaxample `INCAR_final` istead of just `INCAR_4`.

---

targetProperties List of properties to be evaluated in this stage.

Default: `[enthalpy structure]`

Format:

```
targetProperties : [structure energy forces]
```

---

stageType Specifies complex stage type.

Available types:

`execute` — just execute external program using interface

`atomistic` — first assemble atomic structure form components then execute external program using interface

`populationProcessor` — extract set of subsystems from an individual and run its internal stages for each element. Section 5.8.10

Default: atomistic

Format:

```
stageType : populationProcessor
```

5.8.2 VASP interface parameters.

sleepTime

Specifies sleep delay between checks of completion for this stage in seconds.

Default: 30

***Format*:**

```
sleepTime : 2
```

kresol

Specifies the reciprocal-space resolution for k-points generation.

***Format*:**

```
kresol : 0.12
```

Note: Using different values for each step of structure relaxation, starting with cruder (*i.e.*, larger) values and ending with high resolution 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)).

vacuumSize Specify vacuum region size for calculations of nanoparticles, films or surfaces.

Default: 10.0

***Format*:**

```
vacuumSize : 11.0
```

5.8.3 GULP interface parameters.

sleepTime

Specifies sleep delay between checks of completion for this stage in seconds.

Default: 10

***Format*:**

```
sleepTime : 2
```

---

vacuumSize Specify vacuum region size for calculations of nanoparticles, films or surfaces.

Default: 10.0

***Format*:**

```
vacuumSize : 11.0
```

5.8.4 LAMMPS interface parameters.

sleepTime

Specifies sleep delay between checks of completion for this stage in seconds.

Default: 10

***Format*:**

```
sleepTime : 2
```

---

vacuumSize Specify vacuum region size for calculations of nanoparticles, films or surfaces.

Default: 10.0

***Format*:**

```
vacuumSize : 11.0
```

---

mlip

Optional. If provided specifies mlip potential file.

***Format*:**

```
mlip : './Training/La_H_24g_vasp.mtp'
```

5.8.5 Quantum Espresso interface parameters.

sleepTime

Specifies sleep delay between checks of completion for this stage in seconds.

Default: 30

***Format*:**

```
sleepTime : 2
```

kresol

Specifies the reciprocal-space resolution for k-points generation.

***Format*:**

```
kresol : 0.12
```

Note: Using different values for each step of structure relaxation, starting with cruder (*i.e.*, larger) values and ending with high resolution 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)).

vacuumSize Specify vacuum region size for calculations of nanoparticles, films or surfaces.

Default: 10.0

***Format*:**

```
vacuumSize : 11.0
```

5.8.6 Abinit interface parameters.

sleepTime

Specifies sleep delay between checks of completion for this stage in seconds.

Default: 30

***Format*:**

```
sleepTime : 2
```

kresol

Specifies the reciprocal-space resolution for k-points generation.

Default:

***Format*:**

```
kresol : 0.12
```

Note: Using different values for each step of structure relaxation, starting with cruder (*i.e.*, larger) values and ending with high resolution 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)).

vacuumSize Specify vacuum region size for calculations of nanoparticles, films or surfaces.

Default: 10.0

***Format*:**

```
vacuumSize : 11.0
```

5.8.7 FHIaims interface parameters.

sleepTime

Specifies sleep delay between checks of completion for this stage in seconds.

Default: 30

***Format*:**

```
sleepTime : 2
```

---

kresol

Specifies the reciprocal-space resolution for k-points generation.

Default:

***Format*:**

```
kresol : 0.12
```

5.8.8 MOPAC interface parameters.

sleepTime

Specifies sleep delay between checks of completion for this stage in seconds.

Default: 1

***Format*:**

```
sleepTime : 2
```

5.8.9 MLIP interface parameters.

sleepTime

Specifies sleep delay between checks of completion for this stage in seconds.

Default: 10

***Format*:**

```
sleepTime : 2
```

mode

Specifies mlip mode.

Available types:

`select_add` – actively selects configurations to be added to the current training set

`train` – fits an MTP

***Format*:**

```
mode : train
```

potential

Specifies potential file.

***Format*:**

```
potential : './Training/La_H_24g_vasp.mtp'
```

trainingSet

Specifies training set file.

***Format*:**

```
trainingSet : trainingSet: './Training/set.cfg'
```

specorder

Specifies order of species.

***Format*:**

```
specorder : [La H]
```

5.8.10 Population processor parameters.

This is not an interface for third-party program and so it does not provide parameters `commandExecutable`, `taskManager`, `type` and `targetProperties`.

Population processor extracts set of structures from an individual and treats it as a population. For each element it starts nested sequence of stages and controls parallel execution.

stages List of stages to be run on each element.

***Format*:**

```
stages : [vasp_forces]
```

inputKey Specifies name of attribute of an individual to be treated as nested population.

***Format*:**

```
inputKey : trajectory
```

numParallelCalcs Specifies how many structure relaxations you want to run in parallel for this nested loop.

***Format*:**

```
numParallelCalcs : 10
```

5.9 Task manager definition

Task manager definition looks like

```
#define TM
{
type: SBATCH
header: "#!/bin/sh
#SBATCH -p debug
#SBATCH -N 1
#SBATCH -n 1

"
}
```

Type could be:

- SBATCH — for slurm submission system,
- QSUB — for torque submission system,
- BSUB — for lsf submission system.

Header should contain fraction of submission script for supercomputer which you are going to use. Usually such header should contain information on submission queue, number of nodes and cores which the job is going to consume. To determine exact content of such header consult with supercomputer usage guide or administrator.

5.10 Molecules definitions

Each molecule definition looks like

```
#define mol_name
{filename : 'MOL_FILE'}
```

The defined molecule name (like mol_name) should then be used in compositionSpace block (see Section 5.6.1).

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

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

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

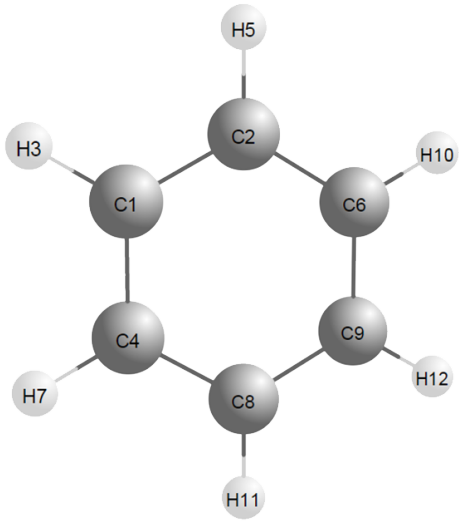

Fig. 5.10.1: Sample of `MOL_FILE` file and illustration of the corresponding molecular structure.

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

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

The 4^{th} atom is C, its coordinates will be set by its distance from the 1^{st} atom, bond angle 4-1-2, and torsion angle 4-1-2-3 — hence, we use "1 2 3" and so forth…until we reach the final, 12^{th} atom, which is H, defined by its distance from the 9^{th} atom (C), bond angle 12-9-6 and torsion angle 12-9-6-8 — hence "9-6-8".

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

5.10.1 How to prepare the `MOL` files

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

5.11 Environment definitions

Environment definition looks like.

```
#define substrate
{
type: substrate
file: './POSCAR_SUBSTRATE'
pbc: (1 1 0)
bufferThickness: 3.0
}
```

The only supported `type` now is `substrate`. Here `file` should contain substrate region in VASP5 POSCAR format, see Fig. 5.11.1.

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

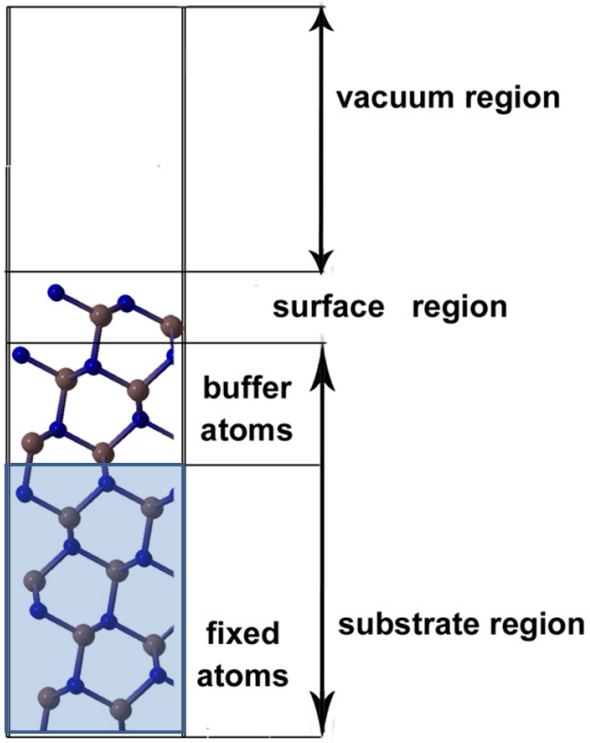

Fig. 5.11.1: **Surface model used in USPEX.**

Besides `file` you need to specify following parameters.

pbc: **(1 1 0)** Periodic boundary conditions. 1 indicates the persistence of periodic boundary conditions and 0 indicates vacuum direction

***Format*:**

```
pbc: (1 1 0)
```

bufferThickness Thickness of the buffer region in substrate. This region is part of `POSCAR_SUBSTRATE`, and is allowed to relax. See Fig. 5.11.1.

***Format*:**

```
bufferThickness:  3.5
```

CHAPTER

SIX

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.

6.1 Structure characterization

Here we have 5 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 `XCRYSDEN` file.

6.2 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.

6.3 Molecular crystals

Here we have 2 utilities:

- MOL precheck — the utility allows you to check files before running USPEX for molecular crystals.
- Zmatrix — the utility converts file to USPEX file.

6.4 Surfaces

Substrate — a program which prepares a substrate from a `POSCAR/CIF` file, given Miller indices, thickness of the layer, and shift. The resulting `POSCAR` file can be used for as a substrate for surface calculations (`dim=2`).

6.5 Miscellaneous

Here we have the following:

- Input generator — USPEX `input.uspex` generator. The utility can help beginners to create a correct input for USPEX calculations.
- Volume estimation — the utility estimates volumes of non-molecular and molecular crystals for USPEX (for `input.uspex` file).
- USPEX manual — online version of this manual.
- USPEX examples — archives with USPEX examples.
- Pressure-composition phase diagram — This program uses the Linear Approximation of Enthalpy (LAE) to calculate the approximate pressure-composition phase diagram from USPEX results from just one pressure.

CHAPTER

SEVEN

APPENDICES

7.1 Block hierarchy

In order to make orientation in parameters and blocks of the `input.uspex` file more visual and convenient, we have prepared an example code template. This template consists of blocks and all parameters or nested blocks of each level.

```
{
   optimizer:
            {
            type: GlobalOptimizer
            target:
                     {
                     type: Atomistic
                     conditions: { externalPressure: <...> }
                     compositionSpace:
                                       {
                                       symbols: [ <...> ]
                                       blocks: [ <...> ]
                                       range: [ <...> ]
                                       minAt: <...>
                                       maxAt: <...>
                                       }
                     ionDistances:
                                 {
                                  < ... >
                                 }
                     cellUtility:
                                 {
                                 dim: < ... >
                                 thickness: < ... >          #if dim=2
                                 axis: ( <...> )
                                 supercellDegree:  <...>     #if dim=2
                                 cellParameters: { <...> }
                                 cellVolume: <...>
                                 cellVectors: <...>
                                 }
                     radialDistributionUtility:
                                                {
                                                Rmax: <...>
                                                delta: <...>
```

(continues on next page)

(continued from previous page)

```
                                        sigma: <...>
                                        tolerance: <...>
                                        }
                    randSym:
                              {
                              nsym: '<...>'
                              splitInto: [ <...> ]
                              }
                    permutation:
                                 {
                                 howManySwaps: <...>
                                 specificSwaps: <...>
                                 }
                    transmutation:
                                 {
                                 howManyTrans: <...>
                                 }
                    softmodemutation:
                                 {
                                 degree: <...>
                                 }
                    randSymPyXtal:
                                 {
                                 nsym: '<...>'
                                 }
                    environmentUtility:
                                       {
                                       environments:  [substrate]
                                       }
                    seeds:
                          {
                          generations: <...>
                          seedsFolders: [ <...> ]
                          }
                    }

          optType: <...>
          selection:
                    {
                    type: USPEXClassic
                    popSize: <...>
                    initialPopSize: <...>
                    bestFrac: <...>
                    howManyDiverse: <...>
                    optType: ( <...> )
                    fractions:
                              {
                              heredity: ( <...> )
                              softModeMutation: ( <...> )
                              randSym: ( <...> )
                              randTop: ( <...> )
                              permutation: ( <...> )
```

(continues on next page)

(continued from previous page)

```
                                randSymPyXtal: ( <...> )
                                }
                        }
                }

    stages: [ <...> ]
    numParallelCalcs: <...>
    numGenerations: <...>
    stopCrit: <...>

    output:
          {
          columns: [ <...>
                   ]
          }
    outputRefreshDelay: <...>
}

#define < external code >
{type : <...>, commandExecutable : '<...>', goptions : ' <...> '}

#define substrate
 {
 type: <...>
 file: '<...>'
 pbc: (<...>)
 bufferThickness: <...>
 }

 #define < Task Manager >
 {
 type: <...>
 header: '<...>'
 }
```

Note: This is just a template with every possible parameter, however, depending on the selected values in parameters or blocks, some of them do not have to be included. Most of the parameters have default values and do not have to be included.

7.2 List of space groups

1	$P1$		2	$P-1$		3	$P2$		4	$P2_1$
5	$C2(A2)*$		6	Pm		7	$Pc(Pa)*$		8	$Cm(Am)*$
9	$Cc(Aa)*$		10	$P2/m$		11	$P2_1/m$		12	$C2/m(a2/m)*$
13	$P2/c(P2/a)*$		14	$P2_1/c(P2_1/a)*$		15	$C2/c(A2/a)*$		16	$P222$
17	$P222_1$		18	$P2_12_12$		19	$P2_12_12_1$		20	$C222_1$
21	$C222$		22	$F222$		23	$I222$		24	$I2_12_12_1$

continues on next page

Table 7.2.1 – continued from previous page

25	$Pmm2$		26	$Pmc2_1$		27	$Pcc2$		28	$Pma2$
29	$Pca2_1$		30	$Pnc2$		31	$Pmn2_1$		32	$Pba2$
33	$Pna2_1$		34	$Pnn2$		35	$Cmm2$		36	$Cmc2_1$
37	$Ccc2$		38	$Amm2(C2mm)*$		39	$Aem2(C2mb)*$		40	$Ama2(C2cm)*$
41	$Aea2(C2cb)*$		42	$Fmm2$		43	$Fdd2$		44	$Imm2$
45	$Iba2$		46	$Ima2$		47	$Pmmm$		48	$Pnnn$
49	$Pccm$		50	$Pban$		51	$Pmma$		52	$Pnna$
53	$Pmna$		54	$Pcca$		55	$Pbam$		56	$Pccn$
57	$Pbcm$		58	$Pnnm$		59	$Pmmn$		60	$Pbcn$
61	$Pbca$		62	$Pnma$		63	$Cmcm$		64	$Cmce(Cmca)*$
65	$Cmmm$		66	$Cccm$		67	$Cmme(Cmma)*$		68	$Ccce(Ccca)*$
69	$Fmmm$		70	$Fddd$		71	$Immm$		72	$Ibam$
73	$Ibca$		74	$Imma$		75	$P4$		76	$P4_1$
77	$P4_2$		78	$P4_3$		79	$I4$		80	$I4_1$
81	$P-4$		82	$I-4$		83	$P4/m$		84	$P42/m$
85	$P4/n$		86	$P4_2/n$		87	$I4/m$		88	$I4_1/a$
89	$P422$		90	$P42_12$		91	$P4_122$		92	$P4_12_12$
93	$P4_222$		94	$P4_22_12$		95	$P4_322$		96	$P4_32_12$
97	$I422$		98	$I4_122$		99	$P4mm$		100	$P4bm$
101	$P4_2cm$		102	$P4_2nm$		103	$P4cc$		104	$P4nc$
105	$P4_2mc$		106	$P4_2bc$		107	$I4mm$		108	$I4cm$
109	$I4_1md$		110	$I4_1cd$		111	$P-42m$		112	$P-42c$
113	$P-42_1m$		114	$P-42_1c$		115	$P-4m2$		116	$P-4c2$
117	$P-4b2$		118	$P-4b2$		119	$I-4m2$		120	$I-4c2$
121	$I-42m$		122	$I-42d$		123	$P4/mmm$		124	$P4/mcc$
125	$P4/nbm$		126	$P4/nnc$		127	$P4/mbm$		128	$P4/mnc$
129	$P4/nmm$		130	$P4/ncc$		131	$P4_2/mmc$		132	$P42/mcm$
133	$P4_2/nbc$		134	$P4_2/nnm$		135	$P4_2/mbc$		136	$P4_2/mnm$
137	$P4_2/nmc$		138	$P4_2/ncm$		139	$I4/mmm$		140	$I4/mcm$
141	$I4_1/amd$		142	$I4_1/acd$		143	$P3$		144	$P3_1$
145	$P3_2$		146	$R3$		147	$P-3$		148	$R-3$
149	$P312$		150	$P321$		151	$P3_112$		152	$P3_121$
153	$P3_212$		154	$P3_221$		155	$R32$		156	$P3m1$
157	$P31m$		158	$P3c1$		159	$P31c$		160	$R3m$
161	$R3c$		162	$P-31m$		163	$P-31c$		164	$P-3m1$
165	$P-3c1$		166	$R-3m$		167	$R-3c$		168	$P6$
169	$P6_1$		170	$P6_5$		171	$P6_2$		172	$P6_4$
173	$P6_3$		174	$P-6$		175	$P6/m$		176	$P6_3/m$
177	$P622$		178	$P6_122$		179	$P6_522$		180	$P6_222$
181	$P6_422$		182	$P6_322$		183	$P6mm$		184	$P6cc$
185	$P6_3cm$		186	$P6_3mc$		187	$P-6m2$		188	$P-6c2$
189	$P-62m$		190	$P-62c$		191	$P6/mmm$		192	$P6/mcc$
193	$P6_3/mcm$		194	$P6_3/mmc$		195	$P23$		196	$F23$
197	$I23$		198	$P2_13$		199	$I2_13$		200	$Pm-3$
201	$Pn-3$		202	$Fm-3$		203	$Fd-3$		204	$Im-3$
205	$Pa-3$		206	$Ia-3$		207	$P432$		208	$P4_232$
209	$F432$		210	$F4_132$		211	$I432$		212	$P4_332$
213	$P4_132$		214	$I4_132$		215	$P-43m$		216	$F-43m$
217	$I-43m$		218	$P-43n$		219	$F-43c$		220	$I-43d$
221	$Pm-3m$		222	$Pn-3n$		223	$Pm-3n$		224	$Pn-3m$

continues on next page

Table 7.2.1 – continued from previous page

225	$P1$		226	$P-1$		227	$P2$		228	$P2_1$
229	$Im-3m$		230	$Ia-3d$						

∗ In parentheses we give 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	$p112/a$						
Monoclinic / orthogonal							
8	$p211$	9	$p2_11$	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	$p2_22$	21	$p2_12_12$	22	$c222$
25	$pmm2$	24	$pma2$	25	$pba2$	26	$cmm2$
27	$pm2m$	28	$pm2_1b$	29	$pm2_1m$	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	$p/4mbn$	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-Maugin	Schönflies	In USPEX
1	C_1	C1 or E
2	C_2	C2
222	D_2	D2
4	C_4	C4
3	C_3	C3
6	C_6	C6
23	T	T
$\bar{1}$	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

continues on next page

Table 7.5.1 – continued from previous page

Hermann-Maugin	Schönflies	In USPEX
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-Maugin	Schönflies	In USPEX
5	C_5	C5
5/m	S_5	S5
$\bar{5}$	S_{10}	S10
5m	Cv_{5v}	Cv5
$\overline{10}$	Ch_{5h}	Ch5
52	D_5	D5
$\bar{5}$m	D_{5d}	Dd5
$\overline{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	C sp^3	0.76	35	Br	1.20	68	Er	1.89
	C sp^2	0.73	36	Kr	1.16	69	Tm	1.90
	C sp	0.69	37	Rb	2.20	70	Yb	1.87

continues on next page

Table 7.6.1 – continued from previous page

Z	Element	radius	Z	Element	radius	Z	Element	radius
7	N	0.71	38	Sr	1.95	71	Lu	1.87
8	O	0.66	39	Y	1.90	72	Hf	1.75
9	F	0.57	40	Zr	1.75	73	Ta	1.70
10	Ne	0.58	41	Nb	1.64	74	W	1.62
11	Na	1.66	42	Mo	1.54	75	Re	1.51
12	Mg	1.41	43	Tc	1.47	76	Os	1.44
13	Al	1.21	44	Ru	1.46	77	Ir	1.41
14	Si	1.11	45	Rh	1.42	78	Pt	1.36
15	P	1.07	46	Pd	1.39	79	Au	1.36
16	S	1.05	47	Ag	1.45	80	Hg	1.32
17	Cl	1.02	48	Cd	1.44	81	Tl	1.45
18	Ar	1.06	49	In	1.42	82	Pb	1.46
19	K	2.03	50	Sn	1.39	83	Bi	1.48
20	Ca	1.76	51	Sb	1.39	84	Po	1.40
21	Sc	1.70	52	Te	1.38	85	At	1.50
22	Ti	1.60	53	I	1.39	86	Rn	1.50
23	V	1.53	54	Xe	1.40	87	Fr	2.60
24	Cr	1.39	55	Cs	2.44	88	Ra	2.21
25	Mn l.s.	1.39	56	Ba	2.15	89	Ac	2.15
	h.s	1.61	57	La	2.07	90	Th	2.06
26	Fe l.s	1.32	58	Ce	2.04	91	Pa	2.00
	h.s.	1.52	59	Pr	2.03	92	U	1.96
27	Co l.s.	1.26	60	Nd	2.01	93	Np	1.90
	h.s.	1.50	61	Pm	1.99	94	Pu	1.87
28	Ni	1.24	62	Sm	1.98	95	Am	1.80
29	Cu	1.32				96	Cm	1.69

Source: Cordero *et al.*, Dalton Trans. 2832-2838, 2008 [22].

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

continues on next page

Table 7.7.1 – continued from previous page

Z	Element	valence	Z	Element	valence	Z	Element	valence
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

continues on next page

Table 7.8.1 – continued from previous page

Z	Element	goodBonds	Z	Element	goodBonds	Z	Element	goodBonds
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. URL: http://www.nature.com/nature/journal/v335/n6187/pdf/335201a0.pdf, doi:doi:10.1038/335201a0.

[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. URL: http://scitation.aip.org/content/aip/journal/jcp/124/24/10.1063/1.2210932, doi:10.1063/1.2210932.

[3] C.W. Glass, A.R. Oganov, and N. Hansen. USPEX — evolutionary crystal structure prediction. *Comp. Phys. Comm.*, 175:713–720, 2006. URL: http://www.sciencedirect.com/science/article/pii/S0010465506002931, doi:10.1016/j.cpc.2006.07.020.

[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. URL: http://dx.doi.org/10.1038/nature02701, doi:10.1038/nature02701.

[5] M. Murakami, K. Hirose, K. Kawamura, N. Sata, and Y. Ohishi. Post-perovskite phase transition in MgSiO3. *Science*, 304(5672):855–858, 2004. URL: http://www.sciencemag.org/content/304/5672/855.abstract, doi:10.1126/science.1095932.

[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. URL: http://onlinelibrary.wiley.com/doi/10.1002/9783527632831.app1/summary, doi:10.1002/9783527632831.app1.

[7] C.J. Pickard and R.J. Needs. High-pressure phases of silane. *Phys. Rev. Lett.*, 97:045504, Jul 2006. URL: http://link.aps.org/doi/10.1103/PhysRevLett.97.045504, doi:10.1103/PhysRevLett.97.045504.

[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. URL: http://link.aps.org/doi/10.1103/PhysRevLett.102.087005, doi:10.1103/PhysRevLett.102.087005.

[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. URL: http://link.aps.org/doi/10.1103/PhysRevLett.102.065501, doi:10.1103/PhysRevLett.102.065501.

[10] C.J. Pickard and R.J. Needs. High-pressure phases of nitrogen. *Phys. Rev. Lett.*, 102:125702, Mar 2009. URL: http://link.aps.org/doi/10.1103/PhysRevLett.102.125702, doi:10.1103/PhysRevLett.102.125702.

[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. URL: http://www.pnas.org/content/107/4/1317.abstract, doi:10.1073/pnas.0908342107.

[12] C.J. Pickard and R.J. Needs. Structures at high pressure from random searching. *physica status solidi (b)*, 246(3):536–540, 2009. URL: http://dx.doi.org/10.1002/pssb.200880546, doi:10.1002/pssb.200880546.

[13] S.T. Call, D.Yu. Zubarev, and A.I. Boldyrev. Global minimum structure searches via particle swarm optimization. *Journal of Computational Chemistry*, 28(7):1177–1186, 2007. URL: http://dx.doi.org/10.1002/jcc.20621, doi:10.1002/jcc.20621.

[14] 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. URL: http://www.sciencedirect.com/science/article/pii/S0010465512004055, doi:10.1016/j.cpc.2012.12.009.

[15] 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. URL: http://www.sciencedirect.com/science/article/pii/S0010465513001392, doi:10.1016/j.cpc.2013.04.004.

[16] 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. URL: http://scitation.aip.org/content/aip/journal/jcp/108/5/10.1063/1.475562, doi:10.1063/1.475562.

[17] S.E. Boulfelfel, A.R. Oganov, and S. Leoni. Understanding the nature of "superhard graphite"'. *Scientific Reports*, 2(471):1–9, 2012. URL: http://www.nature.com/srep/2012/120628/srep00471/full/srep00471.html, doi:10.1038/srep00471.

[18] 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. URL: http://stacks.iop.org/0953-8984/20/i=6/a=064210, doi:doi:10.1088/0953-8984/20/6/064210.

[19] A.R. Oganov and M. Valle. How to quantify energy landscapes of solids. *The Journal of Chemical Physics*, 130:104504, 2009. URL: http://scitation.aip.org/content/aip/journal/jcp/130/10/10.1063/1.3079326, doi:10.1063/1.3079326.

[20] 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. URL: http://pubs.acs.org/doi/abs/10.1021/ar1001318, doi:10.1021/ar1001318.

[21] Pavel V. Bushlanov, Vladislav A. Blatov, and Artem R. Oganov. Topology-based crystal structure generator. *Computer Physics Communications*, 236:1–7, 2019. URL: https://www.sciencedirect.com/science/article/pii/S0010465518303308, doi:https://doi.org/10.1016/j.cpc.2018.09.016.

[22] B. Cordero, V. Gomez, A.E. Platero-Prats, M. Reves, J. Echeverria, E. Cremades, F. Barragan, and S. Alvarez. Covalent radii revisited. *Dalton Trans.*, 21:2832–2838, 2008. URL: http://dx.doi.org/10.1039/B801115J, doi:10.1039/B801115J.