JOURNAL OF APPLIED CRYSTALLOGRAPHY

ISSN: 1600-5767
journals.iucr.org/j

StructureFinder

Daniel Kratzert and Ingo Krossing

J. Appl. Cryst. (2019). **52**, 468–471

JOURNAL OF APPLIED CRYSTALLOGRAPHY
ISSN 1600-5767

StructureFinder

Daniel Kratzert* and Ingo Krossing

Institut für Anorganische und Analytische Chemie, Albert-Ludwigs University of Freiburg, Albertstrasse 21, Freiburg, Baden Württemberg 79104, Germany. *Correspondence e-mail: dkratzert@gmx.de

Received 28 November 2018
Accepted 29 January 2019

Edited by S. Sasaki, Tokyo Institute of Technology, Yokohama, Japan

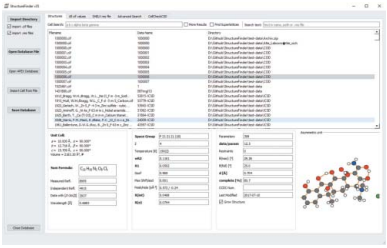
Keywords: single-crystal structures; databases; unit-cell search; CIF; *SHELXL*.

With the program *StructureFinder* it is possible to find single-crystal diffraction measurements stored in any location on a computer. *StructureFinder* collects the unit-cell information and other data from previous measurements and stores them in a database. Searching is performed via a graphical user interface, and both a stand-alone program and a web interface are available.

1. Introduction

Single-crystal structure investigations have been conducted for over a hundred years; however, for the past ten years, the number of measurements and resulting crystal structures has increased dramatically. This is not surprising if one considers that a measurement on a modern diffractometer takes only a few minutes instead of hours, days or even weeks as in former times. However, this increasing amount of data also makes it necessary to properly organize and store the results. This is especially important since measurement, structure solution and refinement have become routine methods and the data are increasingly produced not only by a few experts but more and more by PhD students and postdoctoral researchers. To be able to find these data later, either a rigorous system must be applied in advance or a method has to be used in order to search for structure-refinement results. One solution of the latter class is the Cambridge Structural Database (CSD; Groom *et al.*, 2016), which provides an extremely important and helpful solution to the problem of finding structural data. The OChemDb project focuses on the Crystallographic Open Database and geometry analysis (Altomare *et al.*, 2018; Grazulis *et al.*, 2009). For inorganic structures there is the Inorganic Crystal Structure Database (Belsky *et al.*, 2002). There is also a joint project of several crystallography laboratories called Reciprocal Net. This is a distributed database to store information about molecular structures (http://www.reciprocalnet.org).

But what about the (in-house) structures that are never published, not even as private communications in one of the aforementioned databases? A solution is to collect all of the structure information on a hard disk into a database and provide easy search capabilities for that database. With the program *StructureFinder* it is possible to find single-crystal diffraction measurements stored in any location on a computer. This was also possible with other previous programs. *TEXAN* (Molecular Structure Corporation & Rigaku, 2000), *LCELLS* (Dolomanov *et al.*, 2003, 2009) and *CrysAlisPro* from Rigaku can also search for unit cells in measured structures and there are probably more programs that offer this functionality. However, *StructureFinder* is the



first to have a web interface and the capability to build the database automatically overnight.

2. General implementation

StructureFinder is written in Python3 (https://www.python.org). It uses several software libraries to extend its features. The user front end is implemented with *Qt5* (http://doc.qt.io) for the stand-alone program, and the web interface is implemented with *Bootstrap3* (https://getbootstrap.com), *jQuery* (https://jquery.com), *W2ui* (http://w2ui.com) and *clipboard.js* (https://clipboardjs.com). In both interfaces, molecules are displayed by *JSmol_lite*, a lightweight variant of *JMol/JSmol* (Hanson, 2010). The back end consists of a database interface for *SQLite* (https://www.sqlite.org) in combination with the *bottle.py* (https://bottlepy.org) web framework. The unit-cell search is performed by the cell matching from the *pymatgen* project (Ong *et al.*, 2013).

3. Implementation details

For *StructureFinder* to be useful, the user must first build a database. This can be done either in the graphical user interface or with a command-line program. The program searches recursively for `.cif` (crystallographic information file; Hall *et al.*, 1991) and/or `.res` (*SHELXL* results file; Sheldrick, 2015) files below the given directory.

3.1. Database

All collected file information is first added to a unified CIF-like data structure and stored in an *SQLite* database. The database contains the unit cell, the sum formula, the quality factors of the refinement, file names, full-text search tables and the complete contents of the `.res` file, if applicable.

A different source of structural information for Bruker users is the possibility to open the *APEX2* or *APEX3* database (Bruker, 2018) using the *pg8000* database interface (https://pypi.org/project/pg8000). In this case, the information is limited but a unit-cell search for previous measurements (in which the cell was indexed) is possible.

3.2. Parsers

The `.cif` parser (to read the `.cif` file information) was implemented in pure Python. Previous `.cif` parsers (Brown & McMahon, 2002; Hester, 2006; Gildea *et al.*, 2011; Merkys *et al.*, 2016; Bollinger, 2016; and many others) have not been useful because they were not implemented in Python, were too slow or crashed with faulty files. The `.cif` parser in *StructureFinder* may also be unable to read any obscure `.cif` files, but it is robust and fast enough. In the case of *SHELXL* files, no parsers existed that met the requirements for *StructureFinder*. The *SHELXL* file parser developed for *StructureFinder* went into a separate project and is freely available on GitHub (https://github.com/dkratzert/ShelXFile). Both parsers in *StructureFinder* are optimized for robustness to read even erroneous files as long as there is at least a valid unit cell. Even atoms are not a requirement for successful file parsing.

3.3. Searching

In order to search for a unit cell, two steps are necessary. First, the program must find all unit cells with a similar volume from the database. This step is necessary to avoid comparing all unit cells in the database. Second, the entries in the resulting list of unit cells are compared with the input cell. Because reduction and comparison of unit cells is not trivial, *StructureFinder* takes the lattice-mapping implementation from the lattice module of the excellent *pymatgen* project (Ong *et al.*, 2013; http://pymatgen.org/_modules/pymatgen/core/lattice.html). The literature has plenty of additional examples of how unit cells can be reduced and compared (Andrews & Bernstein, 2014; McGill *et al.*, 2014; and references therein).

3.4. User interface

StructureFinder has two user interfaces. Their functionality is almost the same so here they are described together. The main view shows two important program elements: (i) the list of structures in the database and (ii) the properties (including the molecule view of the currently selected structure; see Fig. 1).

In order to find a crystal structure, the program can search for unit cells. They are put in one single line with space-separated values ($a\ b\ c\ \alpha\ \beta\ \gamma$). This format has the advantage

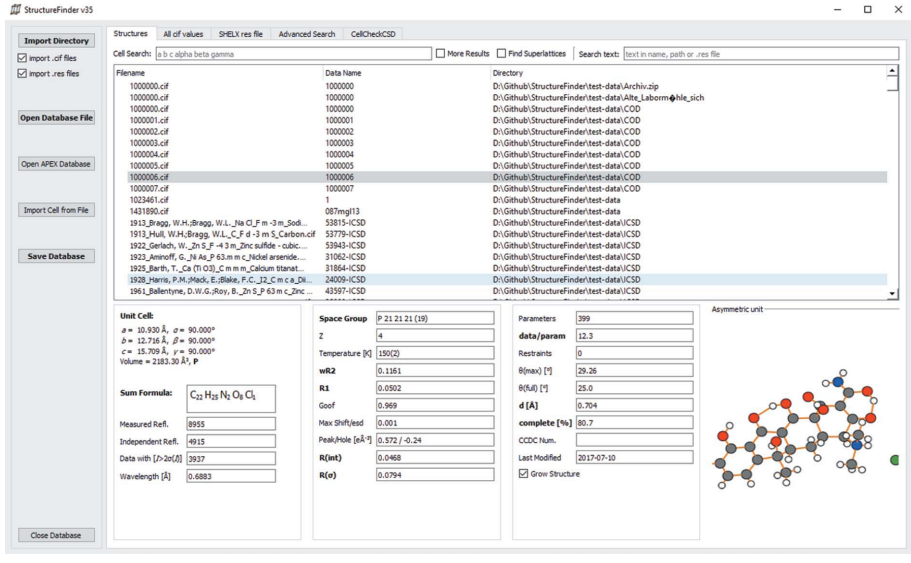


Figure 1
Main window of *StructureFinder*, with a list of structures (top) and residuals (bottom).

that the unit cell can be directly copied from other files such as `.p4p` files, without the need to write each number into a separate field. An even simpler method is to drag and drop a `.cif`, `.res` or `.p4p` file into the window which automatically fills the cell-input field. The unit-cell search has two further options: (i) the search limits can be slackened with the 'more results' option (regular search → volume ±3%, length 0.06 Å, angle 1.0°; more results option → volume ±9%, length 0.2 Å, angle 2.0°) and (ii) the program subsequently searches for supercells with multiples of the cell volume or both options together.

If the unit cell is unknown, there are several other options in the advanced-search tab to find structures. It is possible to search for text strings in the name of the file, the data field of the `.cif` file and the directory name. Importantly, when using *StructureFinder*, the `.res`-file context is also examined during a text search. For example, searching for 'TWIN' finds most of the twinned structures in the database.

The space-group search finds `.cif` files with the desired space group in the case that a space-group number (`_space_group_IT_number`) is specified in the `.cif` file. Some additional very useful features are searching for elements contained in the structure and searching for date ranges to find files modified in a specified date range.

In cases where atoms are found in the files, the program displays the atomic structure of the currently selected database entry (Fig. 1, bottom right). For this task, the extremely fast *JSmol_light* program is employed, meaning that the JavaScript implementation can be run either in a web browser or as a QWebengine widget in the *Qt* interface (https://doc.qt.io/qt-5.11/qwebengineview.html).

At the end, it is possible to save the indexed database from the *Qt* interface in a file and to open it again later.

4. Command-line indexer

For a file server, it is useful to have a regular automatic update of the database without user interaction. Therefore, a command-line indexer `strf_cmd.py` was implemented that can be run every night. For example, the following command indexes `.cif` and `.res` files below two directories and stores the collected data in `mydatabase.sqlite`:

```
python3 ./strf_cmd.py -c -r -d /home/user1 -d
/data/x-ray -o ./mydatabase.sqlite
```

The resulting database file can be opened with both user interfaces.

4.1. Speed

Indexing files in *StructureFinder* is highly optimized; however, most of the time is still spent finding the files to be considered. A whole file server with 30 000 `.res` and `.cif` files in thousands of directories takes about 30 min to index. A smaller directory with 10 000 well ordered files on a solid-state disk can take less than 1 min. But the time depends strongly on the hardware used.

5. Installer

The *StructureFinder* installer executable is only available for Windows 7 onwards (https://www.xs3.uni-freiburg.de/research/structurefinder). However, *StructureFinder* also works on any platform on which Python3 can run if the source code is obtained from GitHub (https://github.com/dkratzert/StructureFinder).

6. Conclusions

StructureFinder has significantly improved the ability to search for 'ancient' structures that have not been archived properly and that are stored anywhere on our group's file server. What used to take hours can now be done in a few moments. In particular, structure refinements that are hidden deep in old user directories can now be found easily. Although this was also possible with other programs such as *TEXAN* (Molecular Structure Corporation & Rigaku, 2000), *LCELLS* (Dolomanov *et al.*, 2003, 2009) and probably more, *StructureFinder* is the first to have a web interface and the capability to build the database automatically overnight. *StructureFinder* is not a replacement for the CSD, but it can be a very helpful addition.

StructureFinder can be obtained free of charge at https://www.xs3.uni-freiburg.de/research/structurefinder or https://github.com/dkratzert/StructureFinder.

Acknowledgements

DK is very grateful to all bug reporters, especially Dieter Schollmeyer, Frank Rominger, Helmar Görls, Hubert Wadepohl, Jürg Hauser, Nils Trapp and others, who provided numerous helpful hints for improvement. We are grateful to the Institute for Inorganic and Analytical Chemistry of the Albert-Ludwigs Universität Freiburg who provided resources and time for this work.

References

Altomare, A., Corriero, N., Cuocci, C., Falcicchio, A., Moliterni, A. & Rizzi, R. (2018). *J. Appl. Cryst.* **51**, 1229–1236.
Andrews, L. C. & Bernstein, H. J. (2014). *J. Appl. Cryst.* **47**, 346–359.
Belsky, A., Hellenbrandt, M., Karen, V. L. & Luksch, P. (2002). *Acta Cryst.* B**58**, 364–369.
Bollinger, J. C. (2016). *J. Appl. Cryst.* **49**, 285–291.
Brown, I. D. & McMahon, B. (2002). *Acta Cryst.* B**58**, 317–324.
Bruker (2018). *APEX2* and *APEX3*. Bruker AXS Inc., Madison, Wisconsin, USA.
Dolomanov, O. V., Blake, A. J., Champness, N. R. & Schröder, M. (2003). *J. Appl. Cryst.* **36**, 955.
Dolomanov, O. V., Bourhis, L. J., Gildea, R. J., Howard, J. A. K. & Puschmann, H. (2009). *J. Appl. Cryst.* **42**, 339–341.
Gildea, R. J., Bourhis, L. J., Dolomanov, O. V., Grosse-Kunstleve, R. W., Puschmann, H., Adams, P. D. & Howard, J. A. K. (2011). *J. Appl. Cryst.* **44**, 1259–1263.
Grazulis, S., Chateigner, D., Downs, R. T., Yokochi, A. F. T., Quirós, M., Lutterotti, L., Manakova, E., Butkus, J., Moeck, P. & Le Bail, A. (2009). *J. Appl. Cryst.* **42**, 726–729.
Groom, C. R., Bruno, I. J., Lightfoot, M. P. & Ward, S. C. (2016). *Acta Cryst.* B**72**, 171–179.

Hall, S. R., Allen, F. H. & Brown, I. D. (1991). *Acta Cryst.* A**47**, 655–685.
Hanson, R. M. (2010). *J. Appl. Cryst.* **43**, 1250–1260.
Hester, J. R. (2006). *J. Appl. Cryst.* **39**, 621–625.
McGill, K. J., Asadi, M., Karakasheva, M. T., Andrews, L. C. & Bernstein, H. J. (2014). *J. Appl. Cryst.* **47**, 360–364.
Merkys, A., Vaitkus, A., Butkus, J., Okulic Kazarinas, M., Kairys, V. & Grazulis, S. (2016). *J. Appl. Cryst.* **49**, 292–301.
Molecular Structure Corporation & Rigaku (2000). *TEXSAN*. MSC, The Woodlands, Texas, USA, and Rigaku Corporation, Tokyo, Japan.
Ong, S. P., Richards, W. D., Jain, A., Hautier, G., Kocher, M., Cholia, S., Gunter, D., Chevrier, V. L., Persson, K. A. & Ceder, G. (2013). *Comput. Mater. Sci.* **68**, 314–319.
Sheldrick, G. M. (2015). *Acta Cryst.* C**71**, 3–8.