Package ‘crystract’

March 20, 2026

Type Package

Title Crystallographic Information File (CIF) Data Processing Tools

Version 1.0.0

Maintainer Don Ngo <dngo@carnegiescience.edu>

Description Provides a suite of functions to parse Crystallographic Information Files (.cif), extracting essential data such as chemical formulas, unit cell parameters, atomic coordinates, and symmetry operations. It also includes tools to calculate interatomic distances, identify bonded pairs using various algorithms (minimum_distance, brunner_nn_reciprocal, econ_nn, crystal_nn), determine nearest neighbor counts, and calculate bond angles. The package is designed to facilitate the preparation of crystallographic data for further analysis, including machine learning applications in materials science.
Methods are described in:
Brunner (1977) <doi:10.1107/S0567739477000461>;
Hoppe (1979) <doi:10.1524/zkri.1979.150.14.23>;
O'Keeffe (1979) <doi:10.1107/S0567739479001765>;
Shannon (1976) <doi:10.1107/S0567739476001551>;
Pan et al. (2021) <doi:10.1021/acs.inorgchem.0c02996>;
Pauling (1960, ISBN:978-0801403330).

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.3.3

Depends R (>= 3.5.0)

Imports data.table, dplyr, stringr, future, future.apply, geometry

Suggests DT, ggplot2, htmlwidgets, knitr, plotly, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

LazyData true

URL https://prabhulab.github.io/ml-crystals/

NeedsCompilation no

Author Don Ngo [aut, cre] (ORCID: <https://orcid.org/0009-0001-2779-2146>),
Anirudh Prabhu [aut] (ORCID: <https://orcid.org/0000-0002-9921-6084>),
Julia-Maria Hubner [aut] (ORCID:
<https://orcid.org/0000-0003-2048-6629>)

Repository CRAN

Date/Publication 2026-03-20 11:50:08 UTC

Contents

`aggregate_batch_results` *Aggregate Batched Analysis Results*

Description

Reads all `batch_*.rds` files from a directory and combines them.

Usage

```
aggregate_batch_results(input_dir, cols_to_keep = NULL)
```

Arguments

`input_dir`	The directory containing RDS files from `analyze_cif_files`.
`cols_to_keep`	Optional character vector of column names to keep.

Value

A single `data.table` containing the aggregated results.

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  out_dir <- file.path(tempdir(), "cif_batches")
  analyze_cif_files(cif_path, output_dir = out_dir)

  aggr <- aggregate_batch_results(out_dir)

  unlink(out_dir, recursive = TRUE)
}
```

`analyze_cif_files` *Analyze a Batch of CIF Files*

Description

A high-level wrapper that analyzes CIF files in batch, supporting parallel processing and batch-to-disk operations.

Usage

```
analyze_cif_files(
  file_paths,
  workers = 1,
  output_dir = NULL,
  batch_size = 1000,
  ...
)
```

Arguments

`file_paths`	Character vector of paths or list of data.tables.
`workers`	Integer. Number of parallel workers.
`output_dir`	Path to output directory (optional).
`batch_size`	Integer.
`...`	Args passed to `analyze_single_cif`.

Value

Data.table or output directory path.

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  out_dir <- file.path(tempdir(), "cif_analysis")
  res <- analyze_cif_files(cif_path, output_dir = out_dir, workers = 1)

  # Clean up
  unlink(out_dir, recursive = TRUE)
}
```

`analyze_single_cif` *Analyze the Content of a Single CIF File*

Description

The core worker function that orchestrates the analysis pipeline for a single crystal structure's data. It dynamically adjusts supercell size and coordinate sets based on the requested bonding algorithms. For geometric algorithms (Voronoi/CrystalNN), it automatically merges atoms occupying the same site to ensure mathematical validity.

Usage

```
analyze_single_cif(
  cif_content,
  file_name = NULL,
  perform_extraction = TRUE,
  perform_calcs_and_transforms = TRUE,
  bonding_algorithms = c("minimum_distance"),
  calculate_bond_angles = TRUE,
  perform_error_propagation = TRUE,
  tolerance = 1e-04,
  minimum_distance_delta = 0.1
)
```

Arguments

`cif_content` Either a `data.table` containing the lines of a CIF file, OR a character string specifying the file path to a CIF file.

`file_name` The name of the original CIF file, used for labeling output.

`perform_extraction`
Logical. If `TRUE`, extracts all metadata.

`perform_calcs_and_transforms`
Logical. If `TRUE`, generates unit cell and supercell.

`bonding_algorithms`
Character vector. Options: `"minimum_distance"`, `"brunner"`, `"econ"`, `"crystal_nn"`, `"voronoi"`.

`calculate_bond_angles`
Logical.

`perform_error_propagation`
Logical.

`tolerance` Numeric. Cutoff for floating-point noise and atom merging (default 1e-4).

`minimum_distance_delta`
Numeric. Tolerance for min-dist algorithm.

Value

A one-row `data.table` with results.

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  res <- analyze_single_cif(cif_path, perform_error_propagation = FALSE)
}
```

apply_symmetry_operations
Apply Symmetry Operations to Generate a Full Unit Cell

Description

Generates all symmetry-equivalent atomic positions within the unit cell.

Usage

```
apply_symmetry_operations(
  atomic_coordinates,
  symmetry_operations,
  unit_cell_metrics,
  tolerance = 1e-04
)
```

Arguments

`atomic_coordinates`
: A `data.table` of asymmetric atoms.

`symmetry_operations`
: A `data.table` of operations.

`unit_cell_metrics`
: A `data.table` of cell parameters (needed for distance merging).

`tolerance`
: Numeric. The distance tolerance in Angstroms for merging close atoms. Default is 1e-4.

Value

A `data.table` containing all unique atomic positions in the unit cell.

See Also

Other coordinate processors: `calculate_expansion_factors()`, `expand_transformed_coords()`

Examples

```
ac <- data.table::data.table(Label = "A", x_a = 0, y_b = 0, z_c = 0)
ops <- data.table::data.table(x = c("x", "-x"), y = c("y", "-y"), z = c("z", "-z"))
uc <- data.table::data.table(`_cell_length_a` = 10, `_cell_length_b` = 10,
                             `_cell_length_c` = 10, `_cell_angle_alpha` = 90,
                             `_cell_angle_beta` = 90, `_cell_angle_gamma` = 90)
apply_symmetry_operations(ac, ops, uc)
```

`brunner_nn_reciprocal` *Identify Atomic Bonds using Brunner's Method (Reciprocal)*

Description

An alternative bonding detection method using the largest reciprocal gap. Matches `pymatgen.analysis.local_env.Brunne`

Usage

```
brunner_nn_reciprocal(distances, tol = 0)
```

Arguments

`distances`	A `data.table` of interatomic distances.
`tol`	A small distance offset (default 0.0) to add to the cutoff.

Value

A `data.table` of bonded pairs.

References

Brunner, G. O. (1977). "A Definition of Coordination and Its Relevance in the Structure Types AlB2 and NiAs." *Acta Crystallographica Section A*, 33(1), 226–227. doi:10.1107/S0567739477000461

See Also

Other bonding algorithms: `crystal_nn()`, `econ_nn()`, `minimum_distance()`, `voronoi_nn()`

Examples

```
dists <- data.table::data.table(Atom1 = c("A", "A"), Atom2 = c("B", "C"),
                                Distance = c(1.5, 2.5),
                                DeltaX = c(1, 0), DeltaY = c(0, 1), DeltaZ = c(0, 0))
brunner_nn_reciprocal(dists)
```

calculate_angles *Calculate Bond Angles*

Description

Calculates all bond angles centered on each atom, formed by pairs of its bonded neighbors.

Usage

```
calculate_angles(
  bonded_pairs,
  atomic_coordinates,
  expanded_coords,
  unit_cell_metrics
)
```

Arguments

`bonded_pairs`	Data.table of bonded atoms.
`atomic_coordinates`	Data.table of asymmetric atom coordinates.
`expanded_coords`	Data.table of supercell atom coordinates.
`unit_cell_metrics`	Data.table with unit cell parameters.

Value

A `data.table` of all unique bond angles.

See Also

Other property calculators: `calculate_distances()`, `calculate_neighbor_counts()`, `calculate_weighted_neighbor_`
`filter_atoms_by_symbol()`, `filter_by_elements()`, `filter_by_wyckoff_symbol()`

Examples

```
bp <- data.table::data.table(Atom1 = c("Si1", "Si1"), Atom2 = c("O1", "O2"))
ac <- data.table::data.table(Label = "Si1", x_a = 0, y_b = 0, z_c = 0)
ec <- data.table::data.table(Label = c("O1", "O2"),
                             x_a = c(1, 0), y_b = c(0, 1), z_c = c(0, 0))
uc <- data.table::data.table(`_cell_length_a` = 10, `_cell_length_b` = 10,
                             `_cell_length_c` = 10, `_cell_angle_alpha` = 90,
                             `_cell_angle_beta` = 90, `_cell_angle_gamma` = 90)
calculate_angles(bp, ac, ec, uc)
```

`calculate_distances` *Calculate Interatomic Distances*

Description

Computes the distances between a central set of atoms and an expanded set, using the metric tensor for accuracy.

Usage

```
calculate_distances(
  atomic_coordinates,
  expanded_coords,
  unit_cell_metrics,
  tolerance = 1e-06
)
```

Arguments

`atomic_coordinates`
: A `data.table` of the primary (asymmetric) atom set.

`expanded_coords`
: A `data.table` of atoms in the expanded supercell.

`unit_cell_metrics`
: A `data.table` with cell parameters.

`tolerance`
: A numeric cutoff (default 1e-6). Distances smaller than this value are considered floating-point noise (overlapping atoms) and are filtered out.

Value

A `data.table` of all non-zero distances.

See Also

Other property calculators: `calculate_angles()`, `calculate_neighbor_counts()`, `calculate_weighted_neighbor_cou` `filter_atoms_by_symbol()`, `filter_by_elements()`, `filter_by_wyckoff_symbol()`

Examples

```
ac <- data.table::data.table(Label = "Si1", x_a = 0, y_b = 0, z_c = 0)
ec <- data.table::data.table(Label = "O1_1", x_a = 0.5, y_b = 0.5, z_c = 0.5)
uc <- data.table::data.table(`_cell_length_a` = 10, `_cell_length_b` = 10,
                             `_cell_length_c` = 10, `_cell_angle_alpha` = 90,
                             `_cell_angle_beta` = 90, `_cell_angle_gamma` = 90)
calculate_distances(ac, ec, uc)
```

calculate_expansion_factors
Calculate Supercell Expansion Factors

Description

Computes the number of unit cell repetitions required in each direction (a, b, c) to encompass a sphere of a given radius.

Usage

```
calculate_expansion_factors(unit_cell_metrics, radius)
```

Arguments

`unit_cell_metrics`
: A `data.table` with cell parameters.

`radius`
: Numeric. The cutoff radius (e.g., 13.0 for Voronoi).

Value

A named numeric vector `c(a=..., b=..., c=...)`.

See Also

Other coordinate processors: `apply_symmetry_operations()`, `expand_transformed_coords()`

Examples

```
uc <- data.table::data.table(`_cell_length_a` = 10, `_cell_length_b` = 10,
                             `_cell_length_c` = 10, `_cell_angle_alpha` = 90,
                             `_cell_angle_beta` = 90, `_cell_angle_gamma` = 90)
calculate_expansion_factors(uc, radius = 13.0)
```

calculate_neighbor_counts
Calculate Coordination Numbers

Description

Counts the number of nearest neighbors for each central atom based on a table of bonded pairs.

Usage

```
calculate_neighbor_counts(bonded_pairs_table)
```

Arguments

bonded_pairs_table
: A `data.table` of bonded pairs.

Value

A `data.table` with columns 'Atom' and 'CoordinationNumber'.

See Also

Other property calculators: `calculate_angles()`, `calculate_distances()`, `calculate_weighted_neighbor_counts()`, `filter_atoms_by_symbol()`, `filter_by_elements()`, `filter_by_wyckoff_symbol()`

Examples

```
bp <- data.table::data.table(Atom1 = c("Si1", "Si1", "O1"),
                             Atom2 = c("O1", "O2", "Si1"))
calculate_neighbor_counts(bp)
```

`calculate_weighted_average_network_distance`
Calculate Weighted Average Network Bond Distance

Description

Computes a single, representative bond length for a specified atomic network. This function precisely implements the validated logic that accounts for site multiplicity and occupancy of the central atoms.

Usage

```
calculate_weighted_average_network_distance(
  distances,
  atomic_coordinates,
  wyckoff_symbols
)
```

Arguments

distances
: A `data.table` of interatomic distances filtered to include **only bonded pairs** (e.g., from `minimum_distance`).

atomic_coordinates
: A `data.table` of asymmetric atoms from `extract_atomic_coordinates`.

wyckoff_symbols
: A character vector of Wyckoff symbols defining the atomic network (e.g., `c("6c", "16i", "24k")`). Must be the full symbol.

Value

A single numeric value representing the weighted average bond distance.

See Also

Other post-processing: filter_ghost_distances(), set_radii_data()

Examples

```
dists <- data.table::data.table(Atom1 = "Si1", Atom2 = "O1", Distance = 1.6)
ac <- data.table::data.table(Label = c("Si1", "O1"),
                             WyckoffMultiplicity = c(4, 4),
                             WyckoffSymbol = c("c", "c"),
                             Occupancy = c(1, 1))
calculate_weighted_average_network_distance(dists, ac, wyckoff_symbols = "4c")
```

---

calculate_weighted_neighbor_counts

Calculate Weighted Coordination Numbers

Description

Counts the weighted number of nearest neighbors for each central atom based on a table of bonded pairs, accounting for the fractional occupancy of the neighbor sites. This is particularly useful for analyzing disordered crystal structures.

If all occupancies are 1.0, this efficiently returns the standard coordination number. If the fractional occupancies on any single crystallographic site sum to greater than 1.0, this indicates an invalid or physically impossible structure; the function will throw a warning and return `NULL` so the file can be discarded by the analysis pipeline.

Usage

```
calculate_weighted_neighbor_counts(bonded_pairs_table, atomic_coordinates)
```

Arguments

`bonded_pairs_table`
: A `data.table` of bonded pairs (e.g., from `minimum_distance`).

`atomic_coordinates`
: A `data.table` of asymmetric atoms from `extract_atomic_coordinates`.

Value

A `data.table` with columns 'Atom', 'CoordinationNumber', and 'WeightedCoordinationNumber'. Returns `NULL` if the occupancies sum to > 1 on any shared site.

See Also

Other property calculators: calculate_angles(), calculate_distances(), calculate_neighbor_counts(), filter_atoms_by_symbol(), filter_by_elements(), filter_by_wyckoff_symbol()

Examples

```
bp <- data.table::data.table(Atom1 = c("Si1", "Si1"),
                             Atom2 = c("O1_1_0_0", "O2_1_0_0"))
ac <- data.table::data.table(Label = c("Si1", "O1", "O2"),
                             Occupancy = c(1.0, 0.5, 0.5),
                             x_a=c(0,0,0), y_b=c(0,0,0), z_c=c(0,0,0))
calculate_weighted_neighbor_counts(bp, ac)
```

---

covalent_radii *Atomic Radii Data for Bond-Length Estimation*

Description

A data.table containing atomic radii for elements. This data is used for estimating plausible bond lengths to filter out non-physical "ghost" distances that can occur in disordered structures. Radii are in Angstroms (Å).

Usage

```
covalent_radii
```

Format

A data table with three columns:

Symbol The chemical symbol of the element.

Radius The atomic radius in Angstroms.

Type The type of radius, e.g., "covalent". The default table only contains covalent radii.

Source

J. Emsley. The Elements. Third edition 1998, Oxford University Press. As provided by Julia-Maria Huebner.

Examples

```
head(covalent_radii)
```

`crystal_nn` *Identify Atomic Bonds using CrystalNN*

Description

Port of Pymatgen's `CrystalNN` weighting logic. Includes Porosity, Electronegativity, and Distance penalties. Uses specific Decision Tree logic for radii (Shannon -> Covalent -> Atomic) per pymatgen's fallback logic.

Usage

```
crystal_nn(
  distances,
  atomic_coordinates,
  expanded_coords,
  unit_cell_metrics,
  cutoff_length = 7,
  x_diff_weight = 3,
  porosity_adjustment = TRUE,
  distance_cutoffs = c(0.5, 1)
)
```

Arguments

`distances`	Ignored.
`atomic_coordinates`	Primary atom set.
`expanded_coords`	Expanded supercell.
`unit_cell_metrics`	Cell parameters.
`cutoff_length`	Voronoi cutoff. Default 7.0 matches Pymatgen CrystalNN default.
`x_diff_weight`	Electronegativity weight (default 3.0).
`porosity_adjustment`	Logical (default TRUE).
`distance_cutoffs`	Vector c(0.5, 1.0).

Value

A `data.table` of bonded pairs.

References

Shannon, R. D. (1976). "Revised effective ionic radii and systematic studies of interatomic distances in halides and chalcogenides." *Acta Crystallographica Section A*, 32(5), 751-767.

Pauling, L. (1960). *The Nature of the Chemical Bond and the Structure of Molecules and Crystals: An Introduction to Modern Structural Chemistry*. Cornell University Press.

Pan, H., Ganose, A. M., Horton, M., et al. (2021). "Benchmarking Coordination Number Prediction Algorithms on Inorganic Crystal Structures." *Inorganic Chemistry*, 60(3), 1590–1603. doi:10.1021/acs.inorgchem.0c02996

See Also

Other bonding algorithms: `brunner_nn_reciprocal()`, `econ_nn()`, `minimum_distance()`, `voronoi_nn()`

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  res <- analyze_single_cif(cif_path, bonding_algorithms = "crystal_nn")
  print(head(res$bonds_crystal_nn[[1]]))
}
```

---

econ_nn — *Identify Atomic Bonds using Hoppe's EconNN Method*

Description

Uses Effective Coordination Numbers (ECoN) and Mean Fictive Ionic Radii (MEFIR).

Usage

```
econ_nn(distances, atomic_coordinates, tol = 0.2, use_fictive_radius = FALSE)
```

Arguments

`distances`	A `data.table` of interatomic distances.
`atomic_coordinates`	A `data.table` of atomic coordinates used to map species to radii.
`tol`	A bond strength cutoff (default 0.2).
`use_fictive_radius`	Logical. If `TRUE`, calculates Hoppe's fictive ionic radius.

Value

A `data.table` of bonded pairs.

References

Hoppe, R. (1979). "Effective Coordination Numbers (ECoN) and Mean Fictive Ionic Radii (MEFIR)." *Zeitschrift Für Kristallographie*, 150(1–4), 23–52. doi:10.1524/zkri.1979.150.14.23

Shannon, R. D. (1976). "Revised Effective Ionic Radii and Systematic Studies of Interatomic Distances in Halides and Chalcogenides." *Acta Crystallographica Section A*, 32(5), 751–767. doi:10.1107/S0567739476001551

See Also

Other bonding algorithms: `brunner_nn_reciprocal()`, `crystal_nn()`, `minimum_distance()`, `voronoi_nn()`

Examples

```
dists <- data.table::data.table(Atom1 = c("Si1", "Si1"), Atom2 = c("O1", "O2"),
                                Distance = c(1.6, 2.0),
                                DeltaX = c(1, 0), DeltaY = c(0, 1), DeltaZ = c(0, 0))
ac <- data.table::data.table(Label = c("Si1", "O1", "O2"),
                             OxidationState = c(4, -2, -2))
econ_nn(dists, ac)
```

`expand_transformed_coords`

Expand Coordinates into a Supercell

Description

Takes a set of atomic coordinates within a single unit cell and replicates them into a supercell grid defined by expansion factors.

Usage

```
expand_transformed_coords(transformed_coords, expansion_factors = c(1, 1, 1))
```

Arguments

`transformed_coords`
: A `data.table` of atom positions within one unit cell.

`expansion_factors`
: A numeric vector `c(a, b, c)` specifying the number of images in each direction (+/-). Default `c(1,1,1)` creates a 3x3x3 grid.

Value

A `data.table` containing all atomic positions in the expanded supercell.

See Also

Other coordinate processors: apply_symmetry_operations(), calculate_expansion_factors()

Examples

```
tc <- data.table::data.table(Label = "A", x_a = 0, y_b = 0, z_c = 0)
expand_transformed_coords(tc, expansion_factors = c(1, 1, 1))
```

---

export_analysis_to_csv

Export Analysis Results to a Directory of CSVs

Description

Exports analysis results to CSV structure.

Usage

```
export_analysis_to_csv(analysis_results, output_dir, overwrite = FALSE)
```

Arguments

analysis_results	A data.table object.
output_dir	Path to main output directory.
overwrite	Logical.

Value

Invisibly returns output_dir.

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  res <- analyze_single_cif(cif_path)

  out_dir <- file.path(tempdir(), "cif_csvs")
  export_analysis_to_csv(res, output_dir = out_dir)

  unlink(out_dir, recursive = TRUE)
}
```

extract_atomic_coordinates

Extract Atomic Coordinates

Description

Parses atomic site info, including labels, fractional coordinates, occupancies, oxidation states, and thermal parameters. It includes logic to extract oxidation states from both site tags and type loops.

Usage

```
extract_atomic_coordinates(cif_content, chemical_formula = NA)
```

Arguments

`cif_content`	A `data.table` containing the lines of a CIF file.
`chemical_formula`	The chemical formula string for validation.

Value

A `data.table` with atomic coordinate data, or `NULL` if not found.

See Also

Other extractors: `extract_chemical_formula()`, `extract_database_code()`, `extract_space_group_name()`, `extract_space_group_number()`, `extract_structure_type()`, `extract_symmetry_operations()`, `extract_unit_cell_metrics()`

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  cif_content <- read_cif_files(cif_path)[[1]]
  extract_atomic_coordinates(cif_content)
}
```

extract_chemical_formula

Extract Chemical Formula from CIF Content

Description

Extracts the chemical sum formula (e.g., from `_chemical_formula_sum`).

Usage

```
extract_chemical_formula(cif_content)
```

Arguments

cif_content A `data.table` containing the lines of a CIF file.

Value

A character string of the chemical formula, or `NA` if not found.

See Also

Other extractors: `extract_atomic_coordinates()`, `extract_database_code()`, `extract_space_group_name()`, `extract_space_group_number()`, `extract_structure_type()`, `extract_symmetry_operations()`, `extract_unit_cell_metrics()`

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  cif_content <- read_cif_files(cif_path)[[1]]
  extract_chemical_formula(cif_content)
}
```

---

extract_database_code *Extract Database Code from CIF Content*

Description

Extracts the database code identifier (e.g., from `_database_code_`) from the CIF.

Usage

```
extract_database_code(cif_content)
```

Arguments

cif_content A `data.table` containing the lines of a CIF file.

Value

A character string of the database code, or `NA` if not found.

See Also

Other extractors: `extract_atomic_coordinates()`, `extract_chemical_formula()`, `extract_space_group_name()`, `extract_space_group_number()`, `extract_structure_type()`, `extract_symmetry_operations()`, `extract_unit_cell_metrics()`

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  cif_content <- read_cif_files(cif_path)[[1]]
  extract_database_code(cif_content)
}
```

---

extract_space_group_name

Extract Space Group Name from CIF Content

Description

Extracts the Hermann-Mauguin space group name (e.g., from `_space_group_name_H-M_alt`).

Usage

```
extract_space_group_name(cif_content)
```

Arguments

`cif_content`	A `data.table` containing the lines of a CIF file.

Value

A character string of the space group name, or `NA` if not found.

See Also

Other extractors: `extract_atomic_coordinates()`, `extract_chemical_formula()`, `extract_database_code()`, `extract_space_group_number()`, `extract_structure_type()`, `extract_symmetry_operations()`, `extract_unit_cell_metrics()`

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  cif_content <- read_cif_files(cif_path)[[1]]
  extract_space_group_name(cif_content)
}
```

`extract_space_group_number`
Extract Space Group Number from CIF Content

Description

Extracts the International Tables space group number (e.g., from `_space_group_IT_number`).

Usage

```
extract_space_group_number(cif_content)
```

Arguments

`cif_content`	A `data.table` containing the lines of a CIF file.

Value

A character string of the space group number, or `NA` if not found.

See Also

Other extractors: `extract_atomic_coordinates()`, `extract_chemical_formula()`, `extract_database_code()`, `extract_space_group_name()`, `extract_structure_type()`, `extract_symmetry_operations()`, `extract_unit_cell_metrics()`

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  cif_content <- read_cif_files(cif_path)[[1]]
  extract_space_group_number(cif_content)
}
```

`extract_structure_type`
Extract Structure Type from CIF Content

Description

Extracts the structure type name (e.g., from `_chemical_name_structure_type`).

Usage

```
extract_structure_type(cif_content)
```

Arguments

`cif_content` A `data.table` containing the lines of a CIF file.

Value

A character string of the structure type, or `NA` if not found.

See Also

Other extractors: `extract_atomic_coordinates()`, `extract_chemical_formula()`, `extract_database_code()`, `extract_space_group_name()`, `extract_space_group_number()`, `extract_symmetry_operations()`, `extract_unit_cell_metrics()`

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  cif_content <- read_cif_files(cif_path)[[1]]
  extract_structure_type(cif_content)
}
```

---

extract_symmetry_operations *Extract Symmetry Operations*

Description

Parses the symmetry operation definitions from the CIF content.

Usage

```
extract_symmetry_operations(cif_content)
```

Arguments

`cif_content` A `data.table` containing the lines of a CIF file.

Value

A `data.table` with symmetry operations. Returns `NULL` if not found.

See Also

Other extractors: `extract_atomic_coordinates()`, `extract_chemical_formula()`, `extract_database_code()`, `extract_space_group_name()`, `extract_space_group_number()`, `extract_structure_type()`, `extract_unit_cell_metrics()`

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  cif_content <- read_cif_files(cif_path)[[1]]
  extract_symmetry_operations(cif_content)
}
```

---

extract_unit_cell_metrics
Extract Unit Cell Metrics

Description

Parses the unit cell parameters (lengths a, b, c and angles alpha, beta, gamma) and their associated standard uncertainties from CIF content.

Usage

```
extract_unit_cell_metrics(cif_content)
```

Arguments

cif_content	A data.table containing the lines of a CIF file.

Value

A one-row data.table with columns for each cell parameter and its error.

See Also

Other extractors: extract_atomic_coordinates(), extract_chemical_formula(), extract_database_code(), extract_space_group_name(), extract_space_group_number(), extract_structure_type(), extract_symmetry_operations()

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  cif_content <- read_cif_files(cif_path)[[1]]
  extract_unit_cell_metrics(cif_content)
}
```

`filter_atoms_by_symbol` *Filter Data by Atom Symbol Interactively*

Description

Prompts the user to select chemical elements to keep in a data table of bonds or angles. Filtering is based on matching the base chemical symbol in a specified column (e.g., "CentralAtom").

Usage

```
filter_atoms_by_symbol(data_table, atom_col = "CentralAtom")
```

Arguments

`data_table`	A `data.table` object containing atomic information, such as the output from `calculate_angles` or `minimum_distance`.
`atom_col`	A character string specifying the name of the column in `data_table` that contains the atom labels to filter by. Defaults to "CentralAtom".

Details

The function first identifies all unique base chemical symbols from the atom labels in the specified column (e.g., it extracts 'C' from 'C1', 'Si' from 'Si2_1'). It then presents these symbols to the user and asks for a comma-separated list of the symbols they wish to retain.

The matching logic is designed to be specific to avoid ambiguity between elements. For example, if the user enters 'C', the function will match labels like 'C1', 'C2', 'C_10', or a lone 'C'. However, it will *not* match labels for different elements that start with C, such as 'Cr1' or 'Ca2'. This is achieved by constructing a regular expression that ensures the character(s) immediately following the selected symbol are not alphabetical letters.

This function is intended for interactive use.

Value

A `data.table` filtered to include only the rows where the atom label in `atom_col` corresponds to one of the user-selected chemical symbols. If the user provides no input, an empty `data.table` is returned.

See Also

Other property calculators: `calculate_angles()`, `calculate_distances()`, `calculate_neighbor_counts()`, `calculate_weighted_neighbor_counts()`, `filter_by_elements()`, `filter_by_wyckoff_symbol()`

Examples

```
# 1. Create a sample data.table of bond angles
sample_angles <- data.table::data.table(
  CentralAtom = c("C1", "C2", "Si1", "Cr1", "O1", "O2", "C"),
  Neighbor1 = c("O1", "O2", "O1", "N1", "C1", "C2", "H1"),
  Neighbor2 = c("H1", "H2", "O2", "N2", "H3", "H4", "H2"),
  Angle = c(109.5, 109.5, 120.0, 90.0, 104.5, 104.5, 120)
)

# 2. In an interactive R session, the function would prompt the user.
if (interactive()) {
  filtered_data <- filter_atoms_by_symbol(sample_angles, atom_col = "CentralAtom")
  print(filtered_data)
}
```

`filter_by_elements` *Filter Distances by Element Symbols*

Description

Removes any distance pair where at least one of the atoms corresponds to a specified list of element symbols.

Usage

```
filter_by_elements(distances, atomic_coordinates, elements_to_exclude)
```

Arguments

`distances`	A `data.table` of interatomic distances.
`atomic_coordinates`	A `data.table` of atomic coordinates.
`elements_to_exclude`	A character vector of element symbols to exclude.

Value

A `data.table` of distances with the specified elements removed.

See Also

Other property calculators: `calculate_angles()`, `calculate_distances()`, `calculate_neighbor_counts()`, `calculate_weighted_neighbor_counts()`, `filter_atoms_by_symbol()`, `filter_by_wyckoff_symbol()`

Examples

```
dists <- data.table::data.table(Atom1 = "Si1_1_0_0", Atom2 = "O1_1_0_0", Distance = 1.6)
ac <- data.table::data.table(Label = c("Si1", "O1"))
filter_by_elements(dists, ac, elements_to_exclude = "O")
```

`filter_by_wyckoff_symbol`

Filter Data by Wyckoff Symbol

Description

Filters a data table (e.g., bonds or angles) to include only entries where a specified atom occupies one of the given Wyckoff sites.

Usage

```
filter_by_wyckoff_symbol(
  data_table,
  atomic_coordinates,
  atom_col,
  wyckoff_symbols
)
```

Arguments

`data_table` A `data.table` object, such as one produced by `minimum_distance` or `calculate_angles`.

`atomic_coordinates` A `data.table` from `extract_atomic_coordinates` containing `WyckoffMultiplicity` and `WyckoffSymbol` columns.

`atom_col` A character string specifying the column in `data_table` that contains the atom labels to filter by (e.g., "Atom1", "CentralAtom").

`wyckoff_symbols` A character vector of the full Wyckoff symbols to keep (e.g., `c("4c")` or `c("6c", "16i", "24k")`).

Details

This function is designed to facilitate analysis based on crystallographic site symmetry. It is particularly useful for analyzing complex structures like clathrates where different atoms perform distinct structural roles based on their site symmetry.

The function works by:

1. Creating a full Wyckoff label (e.g., "4c", "24k") by combining the `WyckoffMultiplicity` and `WyckoffSymbol` columns from the `atomic_coordinates` table.
2. Identifying the parent atom for each entry in the `data_table`.
3. Merging this Wyckoff information into the results table.
4. Filtering to keep only rows where the atom's full Wyckoff label matches one of the symbols provided in the `wyckoff_symbols` vector.

Value

A `data.table` filtered to include only rows where the specified atom occupies one of the desired Wyckoff sites.

See Also

Other property calculators: `calculate_angles()`, `calculate_distances()`, `calculate_neighbor_counts()`, `calculate_weighted_neighbor_counts()`, `filter_atoms_by_symbol()`, `filter_by_elements()`

Examples

```
cif_file <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_file)) {
  # 1. Perform a standard analysis to get bond and coordinate tables
  cif_content <- read_cif_files(cif_file)[[1]]
  atoms <- extract_atomic_coordinates(cif_content)
  metrics <- extract_unit_cell_metrics(cif_content)
  sym_ops <- extract_symmetry_operations(cif_content)
  full_cell <- apply_symmetry_operations(atoms, sym_ops, metrics)
  super_cell <- expand_transformed_coords(full_cell)
  dists <- calculate_distances(atoms, super_cell, metrics)
  bonds <- minimum_distance(dists)

  # 2. Mock Wyckoff sites since 1590946 doesn't explicitly declare them
  atoms[, WyckoffSymbol := "c"]
  atoms[, WyckoffMultiplicity := 4]

  print("Original atomic coordinates showing Wyckoff sites:")
  print(atoms[, .(Label, WyckoffSymbol, WyckoffMultiplicity)])

  filtered_bonds <- filter_by_wyckoff_symbol(
    data_table = bonds,
    atomic_coordinates = atoms,
    atom_col = "Atom1",
    wyckoff_symbols = "4c"
  )

  cat("\nNumber of bonds in original table:", nrow(bonds), "\n")
  cat("Number of bonds after filtering for '4c' site:", nrow(filtered_bonds), "\n")
}
```

`filter_ghost_distances` *Filter Ghost Distances Using Atomic Radii*

Description

Cleans a distance table by removing physically implausible distances. It uses a table of atomic radii to establish a plausible bond length range for each atom pair. Any calculated distance falling outside this range (defined by a `margin`) is considered a "ghost" distance and is removed. This is particularly useful for cleaning data from disordered crystal structures.

Usage

```
filter_ghost_distances(
  distances,
  atomic_coordinates,
  margin = 0.1,
  radii_type = "covalent"
)
```

Arguments

`distances`	A `data.table` of interatomic distances, typically from `calculate_distances`. Must contain `Atom1`, `Atom2`, and `Distance`.
`atomic_coordinates`	A `data.table` of asymmetric atoms from `extract_atomic_coordinates`. Used to link atom labels to element types.
`margin`	A numeric value (default 0.1) specifying the tolerance. A distance `d` between atoms with radii `r1` and `r2` is kept if `(r1+r2)*(1-margin) <= d <= (r1+r2)*(1+margin)`.
`radii_type`	A character string specifying the type of radius to use for the calculation (e.g., `"covalent"`, `"ionic"`). This value must correspond to an entry in the `Type` column of the active radii table. Defaults to `"covalent"`. The radii table can be customized for the session using `set_radii_data()`.

Value

A list containing two `data.tables`:

`kept`	The distances considered physically plausible.
`removed`	The "ghost" distances that were filtered out, with columns explaining the reason for removal.

See Also

Other post-processing: `calculate_weighted_average_network_distance()`, `set_radii_data()`

Examples

```
# Create minimal dummy data for demonstration
distances <- data.table::data.table(
  Atom1 = c("Si1_1_0_0", "O1_1_0_0"),
  Atom2 = c("O1_1_0_0", "Si1_1_0_0"),
  Distance = c(1.6, 0.5) # 0.5 is implausibly short
)
```

```
atomic_coords <- data.table::data.table(
  Label = c("Si1", "O1")
)

# Run the filter
result <- filter_ghost_distances(distances, atomic_coords, margin = 0.1)

print(result$kept)
print(result$removed)
```

---

merge_sites_pbc *Merge Close Atoms (Pymatgen Style)*

Description

Merges atomic sites that are within a specified distance tolerance, accounting for Periodic Boundary Conditions (PBC). This mimics pymatgen's `Structure.merge_sites` logic: it performs hierarchical clustering and then averages the coordinates of the merged sites, taking care of PBC wrapping.

Usage

```
merge_sites_pbc(atomic_coordinates, unit_cell_metrics, tol = 1e-04)
```

Arguments

`atomic_coordinates`
: A data.table with columns `Label`, `x_a`, `y_b`, `z_c`.

`unit_cell_metrics`
: A data.table containing cell lengths and angles.

`tol`
: Numeric. Distance tolerance in Angstroms. Sites closer than this will be merged. Default is 1e-4.

Value

A deduplicated data.table with coordinates averaged.

Examples

```
ac <- data.table::data.table(Label = c("A", "B"),
                             x_a = c(0, 0.00001),
                             y_b = c(0, 0),
                             z_c = c(0, 0))
uc <- data.table::data.table(`_cell_length_a` = 10, `_cell_length_b` = 10,
                             `_cell_length_c` = 10, `_cell_angle_alpha` = 90,
                             `_cell_angle_beta` = 90, `_cell_angle_gamma` = 90)
merge_sites_pbc(ac, uc, tol = 1e-4)
```

`minimum_distance` *Identify Atomic Bonds using the Minimum Distance Method*

Description

Identifies bonded atoms by finding the nearest neighbor distance (d_min) for each central atom and defining a cutoff distance (d_cut) as d_cut = (1 + delta) * d_min.

Usage

```
minimum_distance(distances, delta = 0.1)
```

Arguments

`distances`	A `data.table` of interatomic distances from `calculate_distances`.
`delta`	The relative tolerance parameter (default 0.1).

Value

A `data.table` of bonded pairs.

See Also

Other bonding algorithms: `brunner_nn_reciprocal()`, `crystal_nn()`, `econ_nn()`, `voronoi_nn()`

Examples

```
dists <- data.table::data.table(Atom1 = c("A", "A"), Atom2 = c("B", "C"),
                                Distance = c(1.5, 2.5),
                                DeltaX = c(1, 0), DeltaY = c(0, 1), DeltaZ = c(0, 0))
minimum_distance(dists, delta = 0.1)
```

`propagate_angle_error` *Propagate Angle Error*

Description

Calculates the standard uncertainty for each bond angle.

Usage

```
propagate_angle_error(
  bond_angles,
  atomic_coordinates,
  expanded_coords,
  unit_cell_metrics
)
```

Arguments

bond_angles Data.table of calculated bond angles.

atomic_coordinates Data.table with fractional coordinates and errors.

expanded_coords Data.table of supercell atom coordinates.

unit_cell_metrics Data.table with unit cell parameters and errors.

Value

The input bond_angles data.table with a new 'AngleError' column.

See Also

Other error propagators: propagate_distance_error()

Examples

```
# 1. Create dummy bond angles
ba <- data.table::data.table(
  CentralAtom = "Si1", Neighbor1 = "O1", Neighbor2 = "O2", Angle = 109.5
)

# 2. Create dummy atomic coordinates with errors
ac <- data.table::data.table(
  Label = c("Si1", "O1", "O2"),
  x_a = c(0, 0.1, 0), y_b = c(0, 0, 0.1), z_c = c(0, 0, 0),
  x_error = c(0.01, 0.01, 0.01),
  y_error = c(0.01, 0.01, 0.01),
  z_error = c(0.01, 0.01, 0.01)
)

# 3. Create dummy expanded coordinates
ec <- data.table::data.table(
  Label = c("O1", "O2"),
  x_a = c(0.1, 0), y_b = c(0, 0.1), z_c = c(0, 0)
)

# 4. Create dummy unit cell metrics
uc <- data.table::data.table(
  `_cell_length_a` = 10, `_cell_length_a_error` = 0.1,
  `_cell_length_b` = 10, `_cell_length_b_error` = 0.1,
  `_cell_length_c` = 10, `_cell_length_c_error` = 0.1,
  `_cell_angle_alpha` = 90, `_cell_angle_alpha_error` = 0,
  `_cell_angle_beta` = 90, `_cell_angle_beta_error` = 0,
  `_cell_angle_gamma` = 90, `_cell_angle_gamma_error` = 0
)

# 5. Run the error propagation
propagate_angle_error(ba, ac, ec, uc)
```

propagate_distance_error

Propagate Distance Error

Description

Calculates the standard uncertainty for each interatomic distance.

Usage

```
propagate_distance_error(bonded_pairs, atomic_coordinates, unit_cell_metrics)
```

Arguments

`bonded_pairs`	Data.table of bonded atoms with their distances.
`atomic_coordinates`	Data.table with fractional coordinates and errors.
`unit_cell_metrics`	Data.table with unit cell parameters and errors.

Value

The input `bonded_pairs` data.table with a new 'DistanceError' column.

See Also

Other error propagators: `propagate_angle_error()`

Examples

```
bp <- data.table::data.table(Atom1 = "Si1", Atom2 = "O1_1", Distance = 1.6,
                             DeltaX = 0.1, DeltaY = 0.1, DeltaZ = 0)
ac <- data.table::data.table(Label = c("Si1", "O1"),
                             x_error = c(0.01, 0.01),
                             y_error = c(0.01, 0.01),
                             z_error = c(0.01, 0.01))
uc <- data.table::data.table(`_cell_length_a` = 10, `_cell_length_a_error` = 0.1,
                             `_cell_length_b` = 10, `_cell_length_b_error` = 0.1,
                             `_cell_length_c` = 10, `_cell_length_c_error` = 0.1,
                             `_cell_angle_alpha` = 90, `_cell_angle_alpha_error` = 0,
                             `_cell_angle_beta` = 90, `_cell_angle_beta_error` = 0,
                             `_cell_angle_gamma` = 90, `_cell_angle_gamma_error` = 0)
propagate_distance_error(bp, ac, uc)
```

read_cif_files — *Read CIF Files into Memory*

Description

Reads one or more CIF files from disk and loads each into a `data.table`. It includes encoding sanitization to handle legacy characters.

Usage

```
read_cif_files(file_paths)
```

Arguments

`file_paths`	A character vector of paths to the CIF files.

Value

A list of `data.table` objects.

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  cifs <- read_cif_files(cif_path)
}
```

set_radii_data — *Set or Reset a Custom Atomic Radii Table*

Description

Allows the user to provide their own table of atomic radii for the current R session. This custom table will be used by functions like `filter_ghost_distances`.

Usage

```
set_radii_data(radii_data = NULL)
```

Arguments

`radii_data`	A `data.table` or `data.frame` containing the custom radii data, or `NULL` to reset to the package default.

Details

The provided data.table or data.frame must contain at least three columns:

- Symbol (character): The chemical symbol of the element (e.g., "Si").
- Radius (numeric): The atomic radius in Angstroms (Å).
- Type (character): A descriptor for the radius type (e.g., "covalent", "ionic").

This allows for storing multiple types of radii in the same table, which can be selected using the radii_type argument in relevant functions.

To revert to using the package's default radii table, call the function with NULL or without any arguments.

Value

Invisibly returns NULL. A message is printed to the console confirming the action.

See Also

Other post-processing: calculate_weighted_average_network_distance(), filter_ghost_distances()

Examples

```
# 1. Create a custom radii table with both covalent and ionic radii
my_radii <- data.table::data.table(
  Symbol = c("Si", "O", "O"),
  Radius = c(1.11, 0.66, 1.40),
  Type   = c("covalent", "covalent", "ionic")
)

# 2. Set this table for the current session
set_radii_data(my_radii)

# 3. Reset to the package's default covalent radii table
set_radii_data(NULL)
```

---

voronoi_nn *Identify Atomic Bonds using Voronoi Tessellation*

Description

Performs 3D Voronoi analysis on the supercell.

Usage

```
voronoi_nn(
  atomic_coordinates,
  expanded_coords,
  unit_cell_metrics,
  cutoff = 13,
  tol = 0
)
```

Arguments

`atomic_coordinates`
: A `data.table` of the primary (asymmetric) atom set.

`expanded_coords`
: A `data.table` of atoms in the expanded supercell.

`unit_cell_metrics`
: A `data.table` with cell parameters.

`cutoff`
: Distance cutoff (default 13.0).

`tol`
: Tolerance for solid angle weights (default 0).

Value

A `data.table` of bonded pairs.

References

O'Keeffe, M. (1979). "A Proposed Rigorous Definition of Coordination Number." *Acta Crystallographica Section A*, 35(5), 772–775. doi:10.1107/S0567739479001765

Aurenhammer, F., Klein, R., & Lee, D.-T. (2013). *Voronoi Diagrams and Delaunay Triangulations*. World Scientific Publishing Company.

See Also

Other bonding algorithms: `brunner_nn_reciprocal()`, `crystal_nn()`, `econ_nn()`, `minimum_distance()`

Examples

```
cif_path <- system.file("extdata", "1590946.cif", package = "crystract")
if (file.exists(cif_path)) {
  res <- analyze_single_cif(cif_path, bonding_algorithms = "voronoi")
  print(head(res$bonds_voronoi[[1]]))
}
```

Index