# forte_sets

January 9, 2026

## 1 Forte Sets: Complete PC-Set Universe

This notebook builds a **complete dataset of all 4096 subsets** of the 12 pitch classes {0, 1, …, 11}, along with various **relationship links** between them.

### 1.1 What's Here

#### 1.1.1 Nodes DataFrame (4096 rows)

Each row represents one subset, identified by its **bitmap integer** (0 to 4095). - `id_`: Bitmap integer where bit `i` is set iff pitch class `i` is in the set - `pcset`: Tuple representation, e.g., `(0, 4, 7)` for C major triad - `cardinality`: Number of pitch classes - `prime_form`: Canonical representative under T/I equivalence - `forte_name`: Forte label (e.g., "3-11") if applicable - `interval_vector`: 6-tuple counting interval classes

#### 1.1.2 Link DataFrames

- **Immediate subset links**: Hasse diagram of the subset lattice
- **Complement links**: Each set paired with its complement
- **TI-equivalence links**: Sets sharing the same prime form (set class)
- **Z-relation links**: Sets with same interval vector but different prime form
- **R_p similarity links**: Sets sharing n-1 elements under some T/I

#### 1.1.3 Key Concepts

- **Pitch class (pc)**: Note mod 12 (0=C, 1=C , …, 11=B)
- **Prime form**: Lexicographically smallest form under T_n and I operations
- **Interval vector**: [ic1, ic2, ic3, ic4, ic5, ic6] counts
- **Z-relation**: Same interval content, different prime form
- **Set complex K/Kh**: Reciprocal inclusion relations

[1]:
```
# Import PC-set theory functions from atonal.base
from atonal.base import (
    transpose,
    invert,
    best_normal_order,
    prime_form,
    interval_vector,
    is_transpositionally_symmetric,
```

```
    distinct_transpositions,
    distinct_inversions,
    max_invariance_degrees,
    combinatorial_property_hexachord,
    kh_complex_size,
    z_correspondent_prime_form,
    FORTE_CLASSES,
    PRIME_TO_FORTE,
    FORTE_TO_PRIME,
    forte_name,
    int_to_pcset as _int_to_tuple,
    pcset_to_int as _tuple_to_int,
)

# Note: Use forte_name() directly instead of get_forte_name()
# Note: Use prime_form() for equivalence checking instead of wrappers

print("  Imported PC-set functions from atonal.base")
```

```
  Imported PC-set functions from atonal.base
```

### 1.2 Representation Converter

A universal function to translate between different representations of pc-sets: - **`int`**: Bitmap integer (0 to 4095), where bit `i` is set if pitch class `i` is in the set - **`tuple`**: Sorted tuple of pitch classes, e.g., `(0, 4, 7)` - **`frozenset`** / **`set`**: Unordered collection - **`forte`**: Forte name string, e.g., `"4-19"` (only valid for certain sets) - **`prime`**: Prime form tuple (canonical representative of equivalence class)

[2]:
```
# Import the representation converter from atonal.base
from atonal.base import pc_set_convert, int_to_pcset, pcset_to_int

# Aliases for backward compatibility
_int_to_frozenset = lambda n: frozenset(int_to_pcset(n))

# Test the converter
print("Converter tests:")
print(f"  145 -> tuple: {pc_set_convert(145, 'tuple')}")
print(f"  (0,4,7) -> int: {pc_set_convert((0,4,7), 'int')}")
print(f"  (0,4,7) -> forte: {pc_set_convert((0,4,7), 'forte')}")
print(f"  '3-11' -> prime: {pc_set_convert('3-11', 'prime')}")
```

```
Converter tests:
  145 -> tuple: (0, 4, 7)
  (0,4,7) -> int: 145
  (0,4,7) -> forte: 3-11
  '3-11' -> prime: (0, 3, 7)
```

### 1.3 Nodes DataFrame: All 4096 Subsets

Each row represents one of the $2^{12} = 4096$ possible subsets of $\{0, 1, ..., 11\}$.

The **id** column is the bitmap integer representation.

Additional columns capture properties useful for analysis and filtering.

[3]:
```
import pandas as pd
import numpy as np

# Import the nodes builder from atonal.base (already includes all new fields!)
from atonal.base import build_pcset_nodes_df

# Build the nodes dataframe
nodes_df = build_pcset_nodes_df()
print(f"Nodes DataFrame: {len(nodes_df)} rows")
print(f"\nCardinality distribution:")
print(nodes_df['cardinality'].value_counts().sort_index())
print(f"\nForte sets (prime forms): {nodes_df['is_forte_set'].sum()}")
print(f"Sets with Forte names: {nodes_df['forte_name'].notna().sum()}")


print(f"{nodes_df.shape=}\n")

nodes_df.iloc[30]  # have a look a row
```

```
Nodes DataFrame: 4096 rows

Cardinality distribution:
cardinality
0       1
1      12
2      66
3     220
4     495
5     792
6     924
7     792
8     495
9     220
10     66
11     12
12      1
Name: count, dtype: int64

Forte sets (prime forms): 114
Sets with Forte names: 3662
nodes_df.shape=(4096, 21)
```

[3]:
```
id_                                    30
pcset                          (1, 2, 3, 4)
cardinality                             4
contains_zero                       False
complement_id                        4065
prime_form                     (0, 1, 2, 3)
forte_name                            4-1
is_forte_set                        False
interval_vector            (3, 2, 1, 0, 0, 0)
is_t_symmetric                      False
z_correspondent_prime_form           None
z_correspondent_forte_name           None
n_T                                    12
n_I                                    12
kh_size                                 2
hexachord_combinatorial              None
max_T_invariance                        3
max_T_invariance_n                      1
max_I_invariance                        4
max_I_invariance_n                      5
best_normal_order              (1, 2, 3, 4)
Name: 30, dtype: object
```

[ ]:
```
major_scale_pcset = (0, 2, 4, 5, 7, 9, 11)

# find the index of nodes_df['pcset'] == major_scale_pcset
```

[ ]:
```
np.int64(2741)
```

[9]:
```
t = (nodes_df['pcset'] == major_scale_pcset)
# find the index of the first matching row
index = nodes_df.index[t][0]
index
```

[9]:
```
np.int64(2741)
```

[18]:
```
assert all(nodes_df.id_ == nodes_df.index)  # IDs match index
major_scale_pcset = (0, 2, 4, 5, 7, 9, 11)
major_scale_row = nodes_df[nodes_df['pcset'] == major_scale_pcset].iloc[0]
assert major_scale_row['forte_name'] == '7-35'
# assert major_scale_row['z_correspondent_forte_name'] == '7-35'
# assert major_scale_row['n_T'] == 7
# assert major_scale_row['n_I'] == 7
# assert major_scale_row['kh_size'] == 12
assert major_scale_row['hexachord_combinatorial'] is None
major_scale_row['z_correspondent_forte_name']
```

[19]:
```
major_scale_row
```

[19]:
```
id_                                  2741
pcset                   (0, 2, 4, 5, 7, 9, 11)
cardinality                                 7
contains_zero                            True
complement_id                            1354
prime_form                 (0, 1, 3, 5, 6, 8, 10)
forte_name                                7-35
is_forte_set                             False
interval_vector              (2, 5, 4, 3, 6, 1)
is_t_symmetric                           False
z_correspondent_prime_form                None
z_correspondent_forte_name                None
n_T                                         12
n_I                                         12
kh_size                                      2
hexachord_combinatorial                   None
max_T_invariance                             6
max_T_invariance_n                           5
max_I_invariance                             7
max_I_invariance_n                           4
best_normal_order          (11, 0, 2, 4, 5, 7, 9)
Name: 2741, dtype: object
```

[5]:
```
nodes_df.iloc[40]  # have a look a row
```

[5]:
```
id_                                    40
pcset                                (3, 5)
cardinality                             2
contains_zero                       False
complement_id                        4055
prime_form                          (0, 2)
forte_name                            None
is_forte_set                        False
interval_vector          (0, 1, 0, 0, 0, 0)
is_t_symmetric                      False
z_correspondent_prime_form            None
z_correspondent_forte_name            None
n_T                                     12
n_I                                     12
kh_size                                  2
hexachord_combinatorial               None
max_T_invariance                         1
max_T_invariance_n                       2
max_I_invariance                         2
max_I_invariance_n                       8
best_normal_order                    (3, 5)
Name: 40, dtype: object
```

## 1.4 Link DataFrames: Relationships Between Sets

All link building functions are imported from `atonal.base`. Each link dataframe has columns: - `source`: id of the first set - `target`: id of the second set - (optionally) additional metadata about the relationship

[ ]:
```
# All scalar and boolean predicates have been moved to atonal.base
# This cell is now empty and can be deleted or used for future exploration␣
 ↪utilities

print("  All link building functions are now in atonal.base")
```

[ ]:
```
# Import all link builders from atonal.base
from atonal.base import (
    build_immediate_subset_links_df,
    build_complement_links_df,
    build_ti_equivalence_links_df,
    build_z_relation_links_df,
    build_k_kh_links_df,
    build_rp_similarity_links_df,
)

print("  All link generators imported from atonal.base.")
```

### 1.4.1 Generate Link DataFrames

Let's generate several link dataframes. Note: Some computations are expensive for all 4096×4096 pairs, so we'll use optimizations.

[ ]:
```
# Generate the main link dataframes using imported builders
print("Building link dataframes...")

# 1. Immediate subset links (Hasse diagram of subset lattice)
print("  Building immediate subset links...")
immediate_subset_links = build_immediate_subset_links_df()
print(f"    {len(immediate_subset_links)} edges")

# 2. Complement links
print("  Building complement links...")
complement_links = build_complement_links_df()
print(f"    {len(complement_links)} edges")

# 3. TI-equivalence links (same set class)
print("  Building TI-equivalence links...")
ti_equiv_links = build_ti_equivalence_links_df(nodes_df)
print(f"    {len(ti_equiv_links)} edges")

# 4. Z-relation links
```

```
print("  Building Z-relation links...")
z_relation_links = build_z_relation_links_df(nodes_df)
print(f"    {len(z_relation_links)} edges")

print("\nDone!")
```

### 1.5 Visualization with Force-Directed Graph

Let's visualize the subset lattice using a force-directed graph.

You'll need `cosmograph` for this part. To get it: `pip install cosmograph`

[21]:
```
from cosmograph import cosmo
```

[ ]:
```
# Visualize with cosmograph - the subset lattice Hasse diagram
print("Visualizing subset lattice with cosmograph...")
g1 = cosmo(
    points=nodes_df,
    links=immediate_subset_links,
    point_id_by='id_',
    link_source_by='source',
    link_target_by='target',
    point_size_by='cardinality',
    point_color_by='cardinality',
)
```

```
Visualizing subset lattice with cosmograph…
```

[ ]:
```
Cosmograph(background_color=None, components_display_state_mode=None,
focused_point_ring_color=None, hovered_p…
```

[ ]:
```

```

### 1.6 Additional Link Types

Let's add more sophisticated relationship links.

[ ]:
```
# Import additional link builders from atonal.base
from atonal.base import (
    build_k_kh_links_df,
    build_rp_similarity_links_df,
)

# ------------------------------------------------------------------------
# Build K/Kh and R_p links using imported builders
# ------------------------------------------------------------------------

print("Building K-complex links for sets with cardinality 3-9...")
```

```
forte_range_df = nodes_df[(nodes_df['cardinality'] >= 3) &␣
 ↪(nodes_df['cardinality'] <= 9)]
print(f"  Working with {len(forte_range_df)} sets")

# You can build K or Kh links like this:
# kh_links = build_k_kh_links_df(forte_range_df, kh_only=True)
# k_links = build_k_kh_links_df(forte_range_df, kh_only=False)

print("\nBuilding R_p similarity links for triads (cardinality 3)...")
rp_triads = build_rp_similarity_links_df(nodes_df, cardinality=3)
print(f"  R_p triad links: {len(rp_triads)}")

print("\nBuilding R_p similarity links for tetrads (cardinality 4)...")
rp_tetrads = build_rp_similarity_links_df(nodes_df, cardinality=4)
print(f"  R_p tetrad links: {len(rp_tetrads)}")

print("\n  Link builders imported and used from atonal.base")
```

### 1.7 Summary of Available Data

Let's review what we've built:

[13]:
```
print("=" * 60)
print("NODES DATAFRAME")
print("=" * 60)
print(f"Total rows: {len(nodes_df)}")
print(f"\nColumns: {list(nodes_df.columns)}")
print(f"\nSample rows:")
display(nodes_df[nodes_df['cardinality'].isin([3, 4])].head(10))

print("\n" + "=" * 60)
print("LINK DATAFRAMES")
print("=" * 60)

link_summary = {
    'immediate_subset_links': immediate_subset_links,
    'complement_links': complement_links,
    'ti_equiv_links': ti_equiv_links,
    'z_relation_links': z_relation_links,
    'rp_triads': rp_triads,
    'rp_tetrads': rp_tetrads,
}

for name, df in link_summary.items():
    print(f"\n{name}: {len(df)} edges")
    if len(df) > 0:
        print(f"  Columns: {list(df.columns)}")
```

```
============================================================
NODES DATAFRAME
============================================================
Total rows: 4096

Columns: ['id_', 'pcset', 'cardinality', 'contains_zero', 'complement_id',
'prime_form', 'forte_name', 'is_forte_set', 'interval_vector', 'is_t_symmetric']

Sample rows:
```

| | id_ | pcset | cardinality | contains_zero | complement_id | \ |
|---|---|---|---|---|---|---|
| 7 | 7 | (0, 1, 2) | 3 | True | 4088 | |
| 11 | 11 | (0, 1, 3) | 3 | True | 4084 | |
| 13 | 13 | (0, 2, 3) | 3 | True | 4082 | |
| 14 | 14 | (1, 2, 3) | 3 | False | 4081 | |
| 15 | 15 | (0, 1, 2, 3) | 4 | True | 4080 | |
| 19 | 19 | (0, 1, 4) | 3 | True | 4076 | |
| 21 | 21 | (0, 2, 4) | 3 | True | 4074 | |
| 22 | 22 | (1, 2, 4) | 3 | False | 4073 | |
| 23 | 23 | (0, 1, 2, 4) | 4 | True | 4072 | |
| 25 | 25 | (0, 3, 4) | 3 | True | 4070 | |

| | prime_form | forte_name | is_forte_set | interval_vector | is_t_symmetric |
|---|---|---|---|---|---|
| 7 | (0, 1, 2) | 3-1 | True | (2, 1, 0, 0, 0, 0) | False |
| 11 | (0, 1, 3) | 3-2 | True | (1, 1, 1, 0, 0, 0) | False |
| 13 | (0, 1, 3) | 3-2 | False | (1, 1, 1, 0, 0, 0) | False |
| 14 | (0, 1, 2) | 3-1 | False | (2, 1, 0, 0, 0, 0) | False |
| 15 | (0, 1, 2, 3) | 4-1 | True | (3, 2, 1, 0, 0, 0) | False |
| 19 | (0, 1, 4) | 3-3 | True | (1, 0, 1, 1, 0, 0) | False |
| 21 | (0, 2, 4) | 3-6 | True | (0, 2, 0, 1, 0, 0) | False |
| 22 | (0, 1, 3) | 3-2 | False | (1, 1, 1, 0, 0, 0) | False |
| 23 | (0, 1, 2, 4) | 4-2 | True | (2, 2, 1, 1, 0, 0) | False |
| 25 | (0, 1, 4) | 3-3 | False | (1, 0, 1, 1, 0, 0) | False |

```
============================================================
LINK DATAFRAMES
============================================================

immediate_subset_links: 24576 edges
  Columns: ['source', 'target']

complement_links: 2048 edges
  Columns: ['source', 'target']

ti_equiv_links: 40594 edges
  Columns: ['source', 'target']

z_relation_links: 8796 edges
```

```
  Columns: ['source', 'target', 'interval_vector']

rp_triads: 19872 edges
  Columns: ['source', 'target', 'max_common']

rp_tetrads: 85968 edges
  Columns: ['source', 'target', 'max_common']
```

### 1.8 Interactive Exploration with Cosmograph

Choose different link types to visualize:

[ ]:
```python
# Add human-readable label for visualization
nodes_df['label'] = nodes_df.apply(
    lambda r: f"{r['forte_name'] or ''} {r['pcset']}" if r['cardinality'] <= 6␣
 ↪else str(r['pcset']),
    axis=1
)

# Visualization function
def visualize_links(
    links_df: pd.DataFrame,
    title: str = "PC-Set Network",
    filter_cardinality: tuple = None,
    **cosmo_kwargs
):
    """
    Visualize a link dataframe with cosmograph.

    Args:
        links_df: Links with 'source' and 'target' columns
        title: Title for the visualization
        filter_cardinality: Optional (min, max) to filter nodes
        **cosmo_kwargs: Additional args passed to cosmo()
    """
    # Get nodes involved in these links
    involved_ids = set(links_df['source']) | set(links_df['target'])

    # Filter nodes
    vis_nodes = nodes_df[nodes_df['id_'].isin(involved_ids)].copy()

    if filter_cardinality:
        min_c, max_c = filter_cardinality
        vis_nodes = vis_nodes[(vis_nodes['cardinality'] >= min_c) &␣
 ↪(vis_nodes['cardinality'] <= max_c)]
        involved_ids = set(vis_nodes['id_'])
        links_df = links_df[links_df['source'].isin(involved_ids) &␣
 ↪links_df['target'].isin(involved_ids)]
```

```
    print(f"{title}: {len(vis_nodes)} nodes, {len(links_df)} edges")

    defaults = dict(
        points=vis_nodes,
        links=links_df,
        point_id_by='id_',
        link_source_by='source',
        link_target_by='target',
        point_size_by='cardinality',
        point_color_by='cardinality',
        point_label_by='label',
    )
    defaults.update(cosmo_kwargs)

    return cosmo(**defaults)


# Example visualizations:

print("Available visualizations:")
print("  1. visualize_links(immediate_subset_links, 'Subset Lattice')")
print("  2. visualize_links(ti_equiv_links, 'TI-Equivalence Classes')")
print("  3. visualize_links(complement_links, 'Complement Pairs')")
print("  4. visualize_links(z_relation_links, 'Z-Relations')")
```

```
Available visualizations:
  1. visualize_links(immediate_subset_links, 'Subset Lattice')
  2. visualize_links(ti_equiv_links, 'TI-Equivalence Classes')
  3. visualize_links(complement_links, 'Complement Pairs')
  4. visualize_links(z_relation_links, 'Z-Relations')
```

[17]:
```
g2 = visualize_links(ti_equiv_links, "TI-Equivalence Classes",␣
↪filter_cardinality=(3, 6))
g2
```

```
TI-Equivalence Classes: 2431 nodes, 24493 edges
```

[17]:
```
Cosmograph(background_color=None, components_display_state_mode=None,
focused_point_ring_color=None, hovered_p…
```

### 1.9 Save the data to parquet files

[20]:
```
print(f"{nodes_df.shape=}")
nodes_df.iloc[0]
```

```
nodes_df.shape=(4096, 11)
```

[20]:
```
id_                                0
pcset                             ()
cardinality                        0
contains_zero                  False
complement_id                   4095
prime_form                        ()
forte_name                      None
is_forte_set                   False
interval_vector    (0, 0, 0, 0, 0, 0)
is_t_symmetric                 False
label                             ()
Name: 0, dtype: object
```

[24]:
```
nodes_df.to_parquet('twelve_tone_sets.parquet')
```

[ ]:
```

```