Chemical Structure Searching with Lucene.Net and Indigo

Josh van Eikeren

2015 .NET Fringe Conference

Quick Note

A copy of this presentation and all demo code is available on GitHub:

https://github.com/jvaneikeren/ChemicalStructureSearchingDemo

About Me

Currently CTO/Co-Founder of Avea Solutions

- Create practice management solutions for the mental health industry
- Startup located in Portland, OR
- I work/live remotely in Bend, OR

Previously CTO/Co-Founder of IntelliChem (1998 - 2004) and Blue Reference (2005 – 2012)

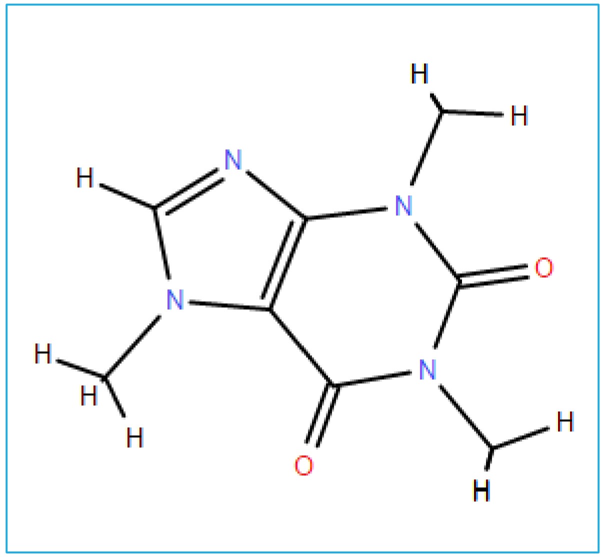
- Built Electronic Laboratory Notebooks (ELNs) for the pharmaceutical industry
- Replaced paper notebooks
- Recorded chemical procedures (“recipes”) as structure data
- Value proposition based on information sharing
- Searching was critically important
  - Text
  - Chemistry objects (chemical structures and reactions)

What is a chemical structure?

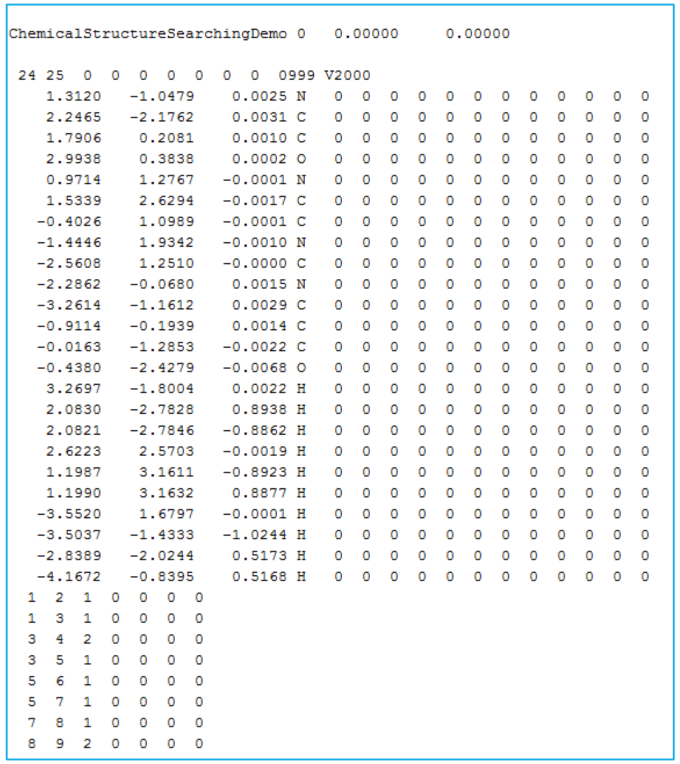
Representation of the atoms and bonds that comprise a molecule

- Connection table (atom/bond connections)
- Coordinates table (visual layout of atoms)

Visually represented as a two-dimensional diagram:



Typically stored in an open-standard ascii file format called a Molfile (*.mol):



```
ChemicalStructureSearchingDemo 0   0.00000     0.00000

 24 25  0  0  0  0  0  0  0  0999 V2000
    1.3120   -1.0479    0.0025 N   0  0  0  0  0  0  0  0  0  0  0  0
    2.2465   -2.1762    0.0031 C   0  0  0  0  0  0  0  0  0  0  0  0
    1.7906    0.2081    0.0010 C   0  0  0  0  0  0  0  0  0  0  0  0
    2.9938    0.3838    0.0002 O   0  0  0  0  0  0  0  0  0  0  0  0
    0.9714    1.2767   -0.0001 N   0  0  0  0  0  0  0  0  0  0  0  0
    1.5339    2.6294   -0.0017 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.4026    1.0989   -0.0001 C   0  0  0  0  0  0  0  0  0  0  0  0
   -1.4446    1.9342   -0.0010 N   0  0  0  0  0  0  0  0  0  0  0  0
   -2.5608    1.2510   -0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
   -2.2862   -0.0680    0.0015 N   0  0  0  0  0  0  0  0  0  0  0  0
   -3.2614   -1.1612    0.0029 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.9114   -0.1939    0.0014 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.0163   -1.2853   -0.0022 C   0  0  0  0  0  0  0  0  0  0  0  0
   -0.4380   -2.4279   -0.0068 O   0  0  0  0  0  0  0  0  0  0  0  0
    3.2697   -1.8004    0.0022 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.0830   -2.7828    0.8938 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.0821   -2.7846   -0.8862 H   0  0  0  0  0  0  0  0  0  0  0  0
    2.6223    2.5703   -0.0019 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.1987    3.1611   -0.8923 H   0  0  0  0  0  0  0  0  0  0  0  0
    1.1990    3.1632    0.8877 H   0  0  0  0  0  0  0  0  0  0  0  0
   -3.5520    1.6797   -0.0001 H   0  0  0  0  0  0  0  0  0  0  0  0
   -3.5037   -1.4333   -1.0244 H   0  0  0  0  0  0  0  0  0  0  0  0
   -2.8389   -2.0244    0.5173 H   0  0  0  0  0  0  0  0  0  0  0  0
   -4.1672   -0.8395    0.5168 H   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0  0  0  0
  1  3  1  0  0  0  0
  3  4  2  0  0  0  0
  3  5  1  0  0  0  0
  5  6  1  0  0  0  0
  5  7  1  0  0  0  0
  7  8  1  0  0  0  0
  8  9  2  0  0  0  0
```

Chemical structure searching importance

Chemical compounds have can have many different names

Example: Prozac

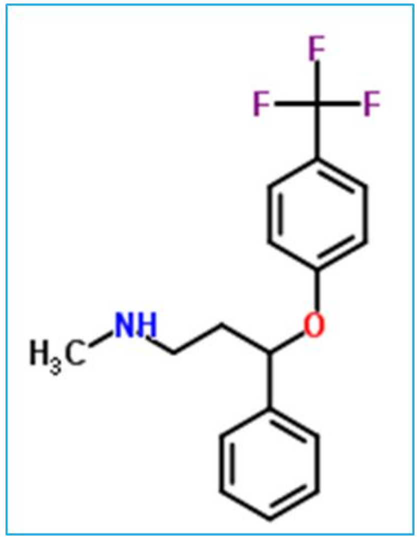
- IUPAC name: N-methyl-3-phenyl-3-[4-(trifluoromethyl)phenoxy]propan-1-amine
- Trade names: Fluoxetine, Deprex
- CAS number: 54910-89-3
- Index identifiers: 39914106

Chemspider lists over **50** text synonyms for Prozac
http://www.chemspider.com/Chemical-Structure.3269.html

Searching by text is not an effective way of locating chemical compound-based relevant information

Chemical compounds, however, have only one chemical structure

Example: Prozac



Historical structure searching solutions

No open source solutions

Proprietary solutions

- Oracle data cartridge (MDL, Accelrys)
  - Molfiles stored in SQL tables
  - Database plug-in (“data cartridge”) performed searching
- Unix server implementation (Daylight)
  - Separate server cluster (required regular data export)

All very expensive

- Example: $100k+ per processor per year

Made product solution pricing difficult; cost had to be passed on to the customer

Modern structure searching solution

Chemical structure searching can be implemented easily in .NET using open source libraries

- Less than 400 lines of code

Lucene.Net

- Embedded document search platform
- Superior search performance through use of an inverted index
- http://lucenenet.apache.org/
- NuGet package: Lucene.Net

Indigo

- Cheminformatics library
- Provides chemical structure manipulation, depiction, and analysis capabilities
- .NET assemblies that wrap C libraries
- http://scitouch.net/opensource/indigo/
- NuGet package: Indigo.Net

Searching approach

Leverage Lucene to perform searching and indexing:

- Lucene employs an inverted index to achieve superior performance
- Lucene is a document search engine that primarily searches text

Must convert the properties of chemical structures into text-based representations which can then be indexed and searched using Lucene

Transforming chemical structures to text for indexing and searching

TEXT

Break text down into tokens

- Example:
  - Text: “The quick brown fox jumps over the lazy dog”
  - Tokens: The, quick, brown, fox, jumps, over, the, lazy, dog

Index tokens

Search index by breaking down query into similar tokens

- Example
  - Search text: “brown fox”
  - Query: brown OR fox

CHEMICAL STRUCTURES

Break chemical structure down into features

- Fixed list of possible features
- Fingerprint: collection of features
- Fixed length byte array
- Byte array index always refers to the same feature
- Structure either has the feature or it doesn’t
- All that matters is the collection of index positions where features have been identified

Index integer fingerprint feature indices as strings

Search index by breaking down query into comparable fingerprint feature index strings

Search types

Exact

- Matches chemical structures with the same composition
- Can’t just compare Molfiles, as structures may be drawn differently
- Instead, generate a unique for each Molfile that is unique based on the connection table

Similarity

- Matches chemical structures that contain similar fingerprint features
- Looking for some overlap between query structure fingerprint features and chemical structure fingerprint features

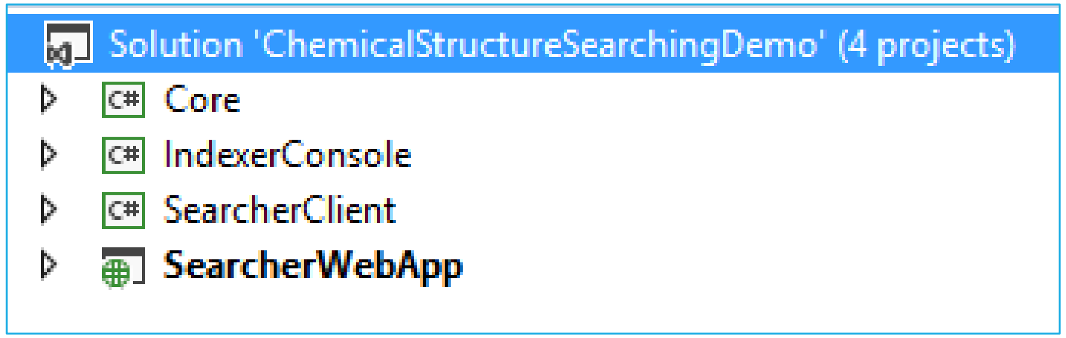
Substructure

- Matches chemical structures that contain a specific substructure
- Looking for complete overlap between query structure fingerprint features and chemical structure fingerprint features
- Must perform an additional analysis to verify actual substructure containment

Searching implementation demo

ChemicalStructureSearchingDemo: Visual Studio 2013 Solution

- **Core**: Library containing the business logic for indexing and searching chemical structures
- **IndexerConsole**: An example console application that indexes a directory of Molfiles
- **SearcherClient**: A WinForms application that can search an index of chemical structures
- **SearcherWebApp**: An ASP.NET MVC web application that can search an index of chemical structures



Solution 'ChemicalStructureSearchingDemo' (4 projects)
- Core
- IndexerConsole
- SearcherClient
- SearcherWebApp

Core object model

ChemicalStructure.cs

- Encapsulates the contents of a chemical structure (i.e., Molfile)
- Uses Indigo to generate a unique key, fingerprints, and images for the chemical structure

ChemicalStructureIndexer.cs

- Uses Lucene.Net to create a Lucene index of chemical structures

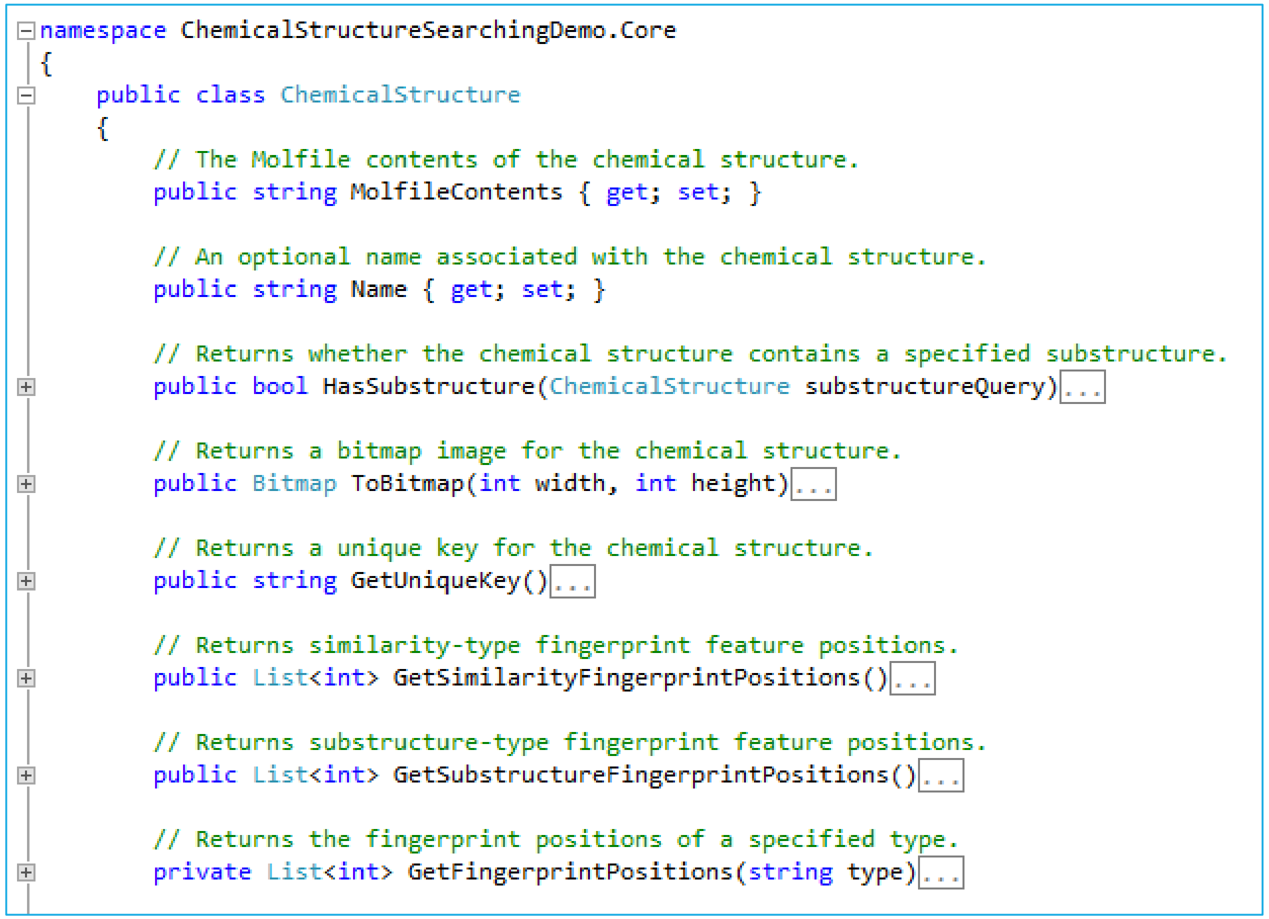
ChemicalStructureSearcher.cs

- Uses Lucene.Net to search a Lucene index of chemical structures

ChemicalStructureSearchResult.cs

- Encapsulates a search match
- Includes the score for the search match as well as the associated chemical structure

ChemicalStructure.cs



```
namespace ChemicalStructureSearchingDemo.Core
{
    public class ChemicalStructure
    {
        // The Molfile contents of the chemical structure.
        public string MolfileContents { get; set; }

        // An optional name associated with the chemical structure.
        public string Name { get; set; }

        // Returns whether the chemical structure contains a specified substructure.
        public bool HasSubstructure(ChemicalStructure substructureQuery)...

        // Returns a bitmap image for the chemical structure.
        public Bitmap ToBitmap(int width, int height)...

        // Returns a unique key for the chemical structure.
        public string GetUniqueKey()...

        // Returns similarity-type fingerprint feature positions.
        public List<int> GetSimilarityFingerprintPositions()...

        // Returns substructure-type fingerprint feature positions.
        public List<int> GetSubstructureFingerprintPositions()...

        // Returns the fingerprint positions of a specified type.
        private List<int> GetFingerprintPositions(string type)...
```

Fingerprint generation

Indigo is used to generate a unique key

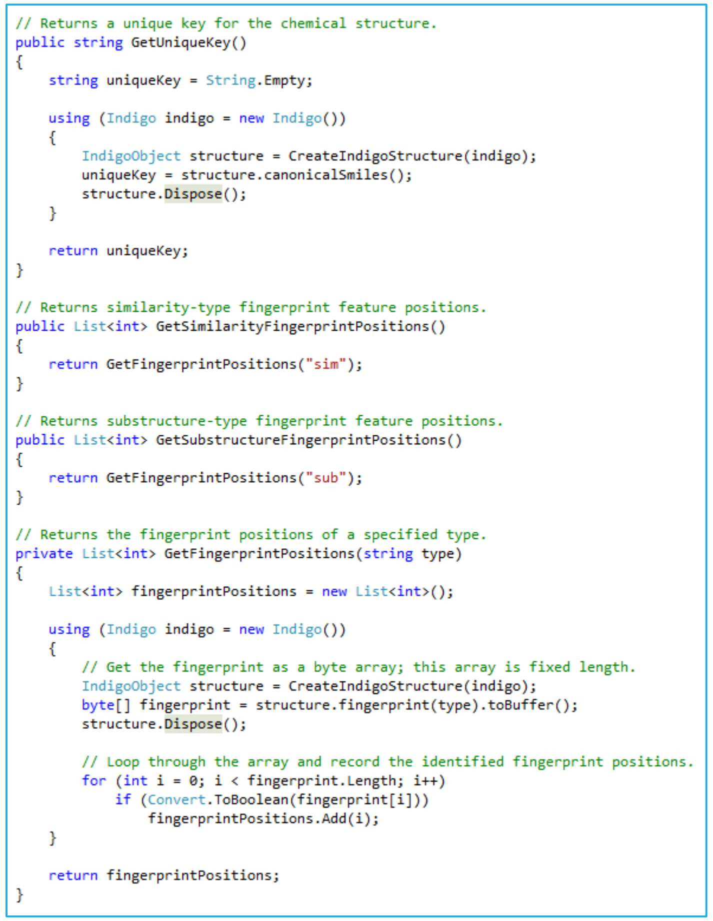
- Canonical smiles string

Indigo is used to generate two different fingerprint types

- Similarity
- Substructure

Fingerprints

- Returned as a fixed-length byte array
- Each index position corresponds to the same feature across all structures
- Every structure either has a feature or it doesn’t; hence a byte value of 1 or 0 for the feature
- Can be condensed into a list of integers comprising the identified feature index positions



```
// Returns a unique key for the chemical structure.
public string GetUniqueKey()
{
    string uniqueKey = String.Empty;

    using (Indigo indigo = new Indigo())
    {
        IndigoObject structure = CreateIndigoStructure(indigo);
        uniqueKey = structure.canonicalSmiles();
        structure.Dispose();
    }

    return uniqueKey;
}

// Returns similarity-type fingerprint feature positions.
public List<int> GetSimilarityFingerprintPositions()
{
    return GetFingerprintPositions("sim");
}

// Returns substructure-type fingerprint feature positions.
public List<int> GetSubstructureFingerprintPositions()
{
    return GetFingerprintPositions("sub");
}

// Returns the fingerprint positions of a specified type.
private List<int> GetFingerprintPositions(string type)
{
    List<int> fingerprintPositions = new List<int>();

    using (Indigo indigo = new Indigo())
    {
        // Get the fingerprint as a byte array; this array is fixed length.
        IndigoObject structure = CreateIndigoStructure(indigo);
        byte[] fingerprint = structure.fingerprint(type).toBuffer();
        structure.Dispose();

        // Loop through the array and record the identified fingerprint positions.
        for (int i = 0; i < fingerprint.Length; i++)
            if (Convert.ToBoolean(fingerprint[i]))
                fingerprintPositions.Add(i);
    }

    return fingerprintPositions;
}
```

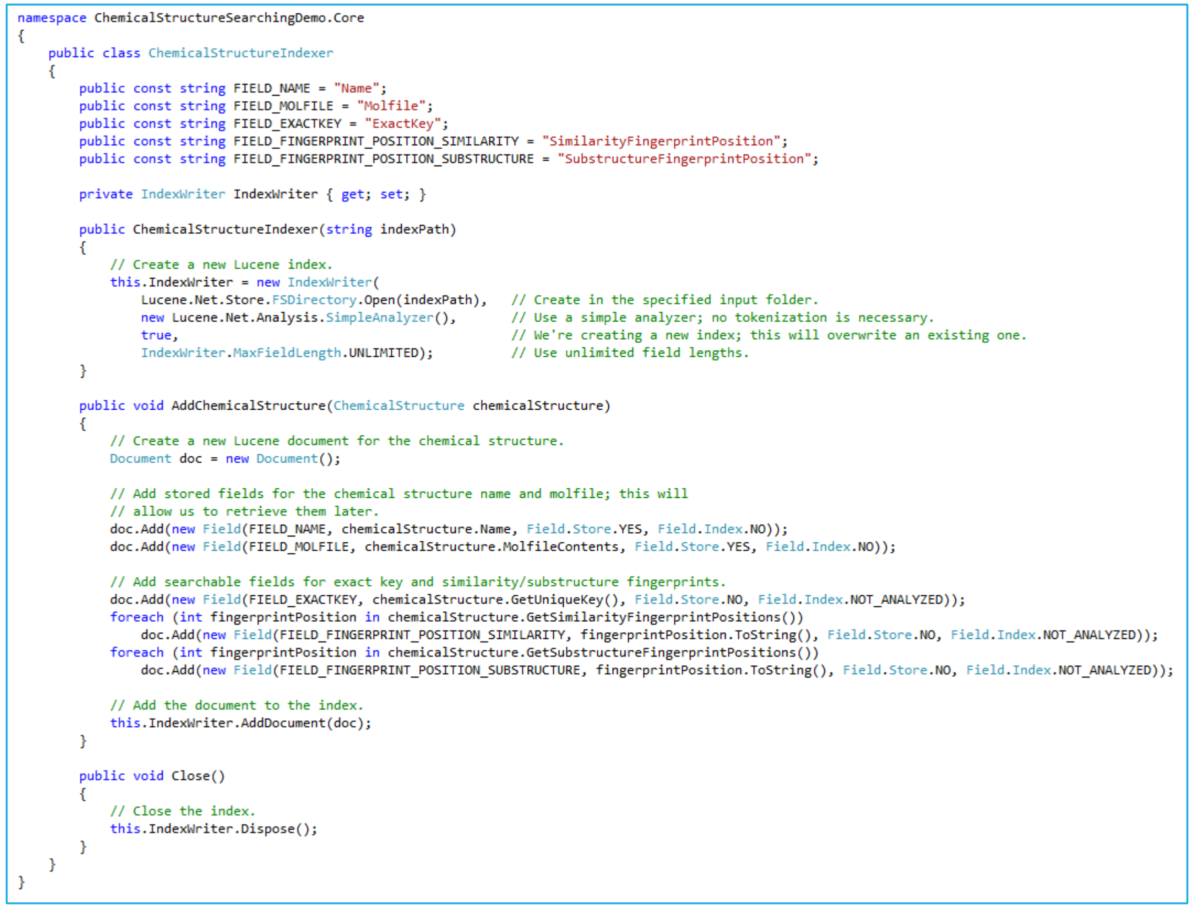
ChemicalStructureIndexer.cs

Creates a new Lucene index

Indexes added chemical structures

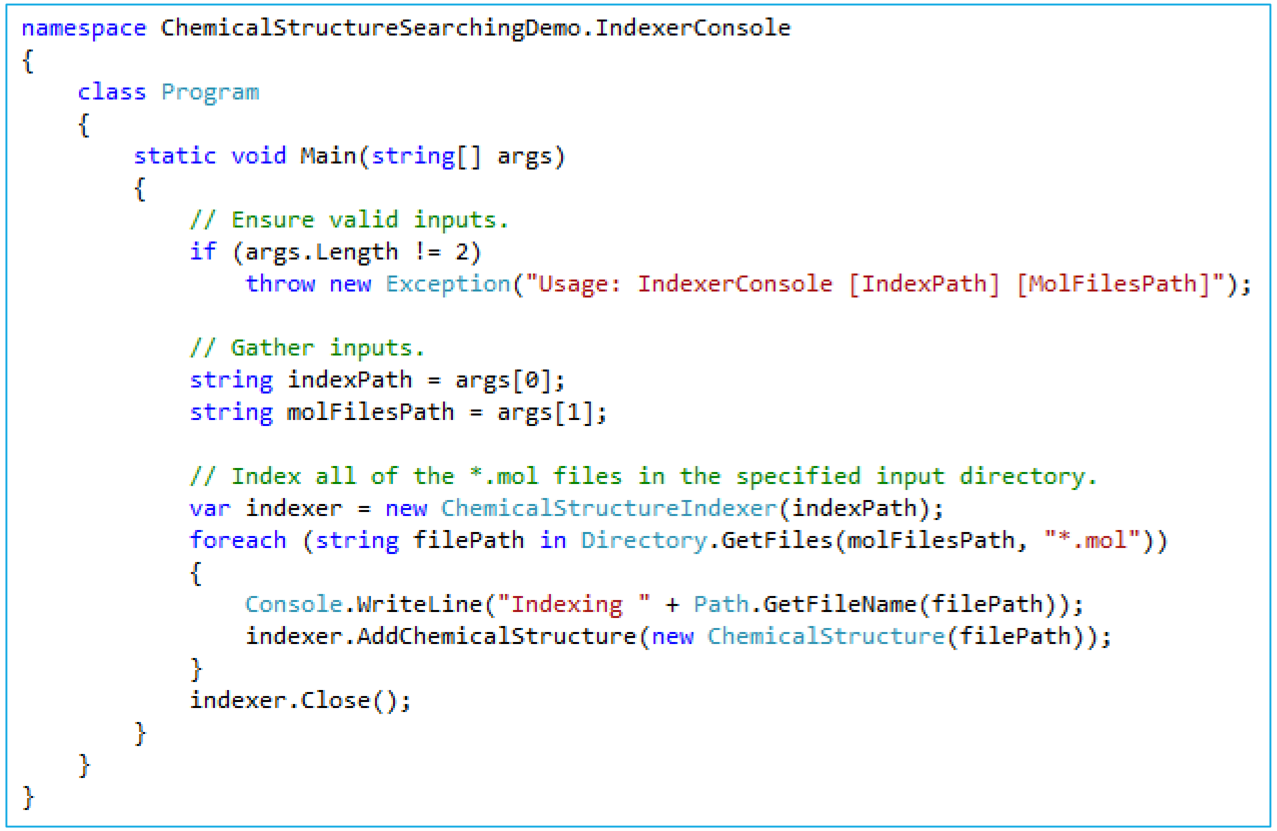
- Adds Name and MolfileContents as retrievable fields
- Adds unique key, similarity feature positions, and substructure feature positions as searchable fields

Key: All fingerprint elements have been converted into a collection of searchable strings



```
namespace ChemicalStructureSearchingDemo.Core
{
    public class ChemicalStructureIndexer
    {
        public const string FIELD_NAME = "Name";
        public const string FIELD_MOLFILE = "Molfile";
        public const string FIELD_EXACTKEY = "ExactKey";
        public const string FIELD_FINGERPRINT_POSITION_SIMILARITY = "SimilarityFingerprintPosition";
        public const string FIELD_FINGERPRINT_POSITION_SUBSTRUCTURE = "SubstructureFingerprintPosition";

        private IndexWriter IndexWriter { get; set; }

        public ChemicalStructureIndexer(string indexPath)
        {
            // Create a new Lucene index.
            this.IndexWriter = new IndexWriter(
                Lucene.Net.Store.FSDirectory.Open(indexPath),   // Create in the specified input folder.
                new Lucene.Net.Analysis.SimpleAnalyzer(),       // Use a simple analyzer; no tokenization is necessary.
                true,                                           // We're creating a new index; this will overwrite an existing one.
                IndexWriter.MaxFieldLength.UNLIMITED);          // Use unlimited field lengths.
        }

        public void AddChemicalStructure(ChemicalStructure chemicalStructure)
        {
            // Create a new Lucene document for the chemical structure.
            Document doc = new Document();

            // Add stored fields for the chemical structure name and molfile; this will
            // allow us to retrieve them later.
            doc.Add(new Field(FIELD_NAME, chemicalStructure.Name, Field.Store.YES, Field.Index.NO));
            doc.Add(new Field(FIELD_MOLFILE, chemicalStructure.MolfileContents, Field.Store.YES, Field.Index.NO));

            // Add searchable fields for exact key and similarity/substructure fingerprints.
            doc.Add(new Field(FIELD_EXACTKEY, chemicalStructure.GetUniqueKey(), Field.Store.NO, Field.Index.NOT_ANALYZED));
            foreach (int fingerprintPosition in chemicalStructure.GetSimilarityFingerprintPositions())
                doc.Add(new Field(FIELD_FINGERPRINT_POSITION_SIMILARITY, fingerprintPosition.ToString(), Field.Store.NO, Field.Index.NOT_ANALYZED));
            foreach (int fingerprintPosition in chemicalStructure.GetSubstructureFingerprintPositions())
                doc.Add(new Field(FIELD_FINGERPRINT_POSITION_SUBSTRUCTURE, fingerprintPosition.ToString(), Field.Store.NO, Field.Index.NOT_ANALYZED));

            // Add the document to the index.
            this.IndexWriter.AddDocument(doc);
        }

        public void Close()
        {
            // Close the index.
            this.IndexWriter.Dispose();
        }
    }
}
```

Indexing example: IndexerConsole.exe



```
namespace ChemicalStructureSearchingDemo.IndexerConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            // Ensure valid inputs.
            if (args.Length != 2)
                throw new Exception("Usage: IndexerConsole [IndexPath] [MolFilesPath]");

            // Gather inputs.
            string indexPath = args[0];
            string molFilesPath = args[1];

            // Index all of the *.mol files in the specified input directory.
            var indexer = new ChemicalStructureIndexer(indexPath);
            foreach (string filePath in Directory.GetFiles(molFilesPath, "*.mol"))
            {
                Console.WriteLine("Indexing " + Path.GetFileName(filePath));
                indexer.AddChemicalStructure(new ChemicalStructure(filePath));
            }
            indexer.Close();
        }
    }
}
```

ChemicalStructureSearcher.cs

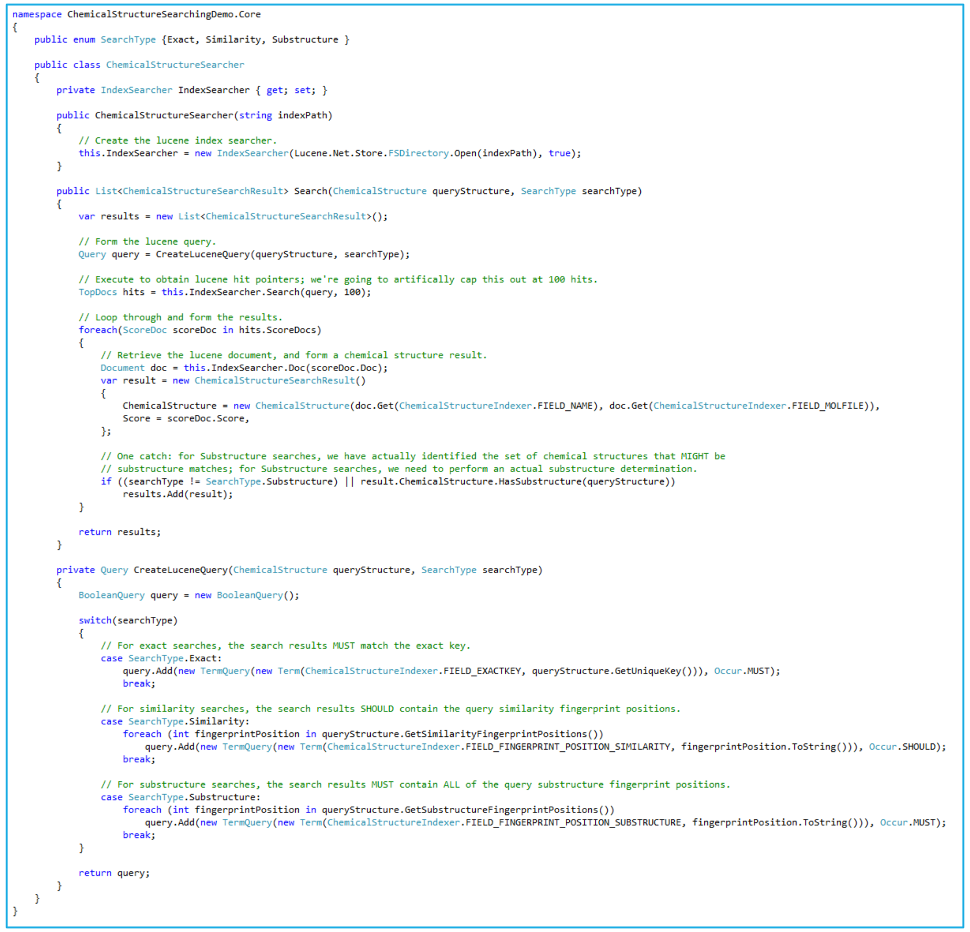
Searches the Lucene index

Takes a query structure and search type as arguments

Builds a Lucene query as such:

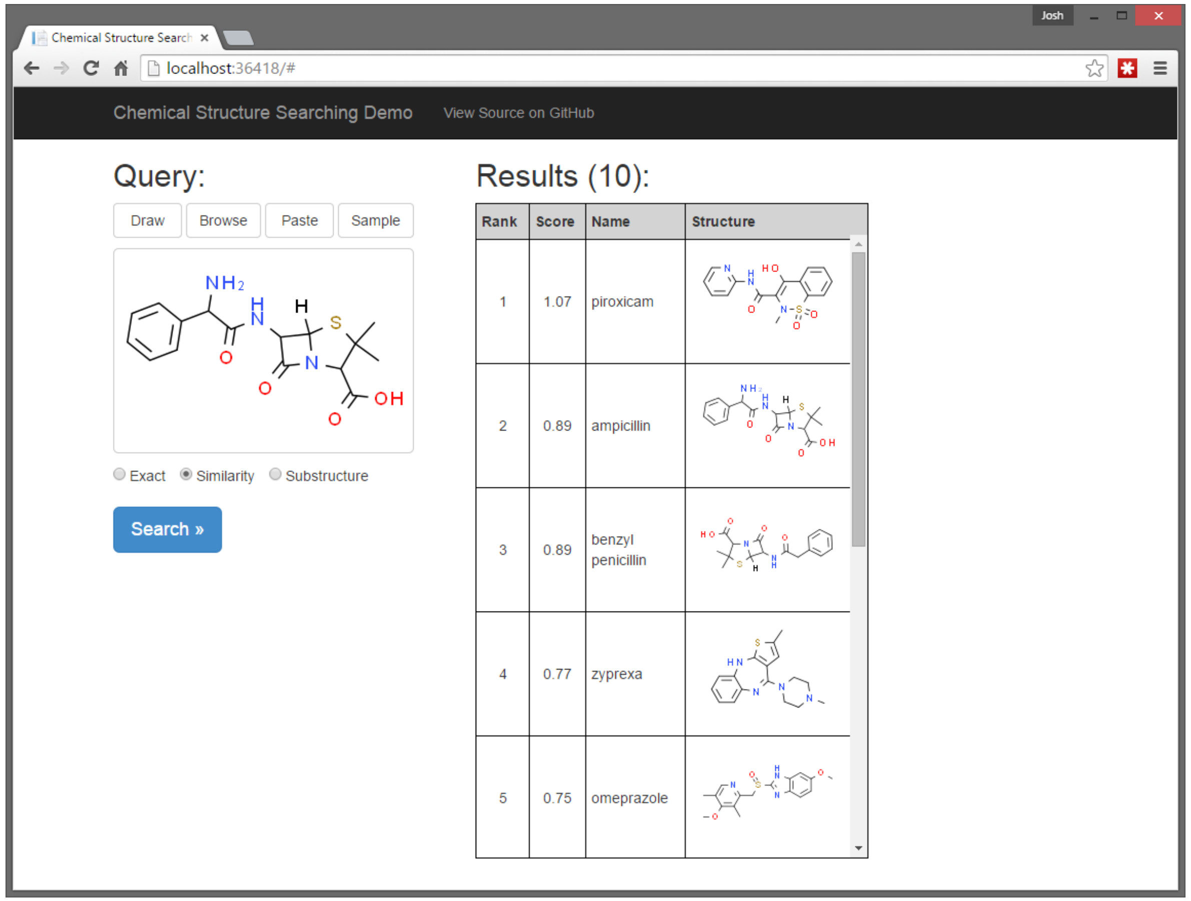
- For the Exact search type, we search for an exact match on the query structure's unique key
- For the Similarity search type, we search for matches between the query structure's similarity fingerprint positions; the more matches, the higher the score
- For the Substructure search type, we search for an exact match on the query structure's substructure fingerprint positions
  - These hits, however, only represent possible matches
  - A secondary analysis must then be performed to determine if the match actually indeed does contain the query substructure

Extracts name and Molfile contents to return as ChemicalStructureSearchResult objects



```
namespace ChemicalStructureSearchingDemo.Core
{
    public enum SearchType {Exact, Similarity, Substructure }

    public class ChemicalStructureSearcher
    {
        private IndexSearcher IndexSearcher { get; set; }

        public ChemicalStructureSearcher(string indexPath)
        {
            // Create the lucene index searcher.
            this.IndexSearcher = new IndexSearcher(Lucene.Net.Store.FSDirectory.Open(indexPath), true);
        }

        public List<ChemicalStructureSearchResult> Search(ChemicalStructure queryStructure, SearchType searchType)
        {
            var results = new List<ChemicalStructureSearchResult>();

            // Form the lucene query.
            Query query = CreateLuceneQuery(queryStructure, searchType);

            // Execute to obtain lucene hit pointers; we're going to artifically cap this out at 100 hits.
            TopDocs hits = this.IndexSearcher.Search(query, 100);

            // Loop through and form the results.
            foreach(ScoreDoc scoreDoc in hits.ScoreDocs)
            {
                // Retrieve the lucene document, and form a chemical structure result.
                Document doc = this.IndexSearcher.Doc(scoreDoc.Doc);
                var result = new ChemicalStructureSearchResult()
                {
                    ChemicalStructure = new ChemicalStructure(doc.Get(ChemicalStructureIndexer.FIELD_NAME), doc.Get(ChemicalStructureIndexer.FIELD_MOLFILE)),
                    Score = scoreDoc.Score,
                };

                // One catch: for Substructure searches, we have actually identified the set of chemical structures that MIGHT be
                // substructure matches; for Substructure searches, we need to perform an actual substructure determination.
                if ((searchType != SearchType.Substructure) || result.ChemicalStructure.HasSubstructure(queryStructure))
                    results.Add(result);
            }

            return results;
        }

        private Query CreateLuceneQuery(ChemicalStructure queryStructure, SearchType searchType)
        {
            BooleanQuery query = new BooleanQuery();

            switch(searchType)
            {
                // For exact searches, the search results MUST match the exact key.
                case SearchType.Exact:
                    query.Add(new TermQuery(new Term(ChemicalStructureIndexer.FIELD_EXACTKEY, queryStructure.GetUniqueKey())), Occur.MUST);
                    break;

                // For similarity searches, the search results SHOULD contain the query similarity fingerprint positions.
                case SearchType.Similarity:
                    foreach (int fingerprintPosition in queryStructure.GetSimilarityFingerprintPositions())
                        query.Add(new TermQuery(new Term(ChemicalStructureIndexer.FIELD_FINGERPRINT_POSITION_SIMILARITY, fingerprintPosition.ToString())), Occur.SHOULD);
                    break;

                // For substructure searches, the search results MUST contain ALL of the query substructure fingerprint positions.
                case SearchType.Substructure:
                    foreach (int fingerprintPosition in queryStructure.GetSubstructureFingerprintPositions())
                        query.Add(new TermQuery(new Term(ChemicalStructureIndexer.FIELD_FINGERPRINT_POSITION_SUBSTRUCTURE, fingerprintPosition.ToString())), Occur.MUST);
                    break;
            }

            return query;
        }
    }
}
```

Searching Demo: SearcherWebApp



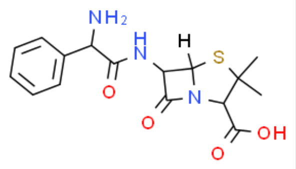
Chemical Structure Search

localhost:36418/#

Chemical Structure Searching Demo   View Source on GitHub

Query:

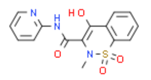
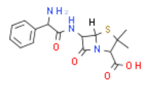
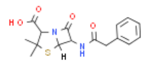
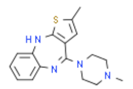
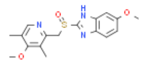
Draw Browse Paste Sample



( ) Exact [✓ Similarity] ( ) Substructure

Search »

Results (10):

Rank	Score	Name	Structure
1	1.07	piroxicam	
2	0.89	ampicillin	
3	0.89	benzyl penicillin	
4	0.77	zyprexa	
5	0.75	omeprazole	

Questions – AMA!

A copy of this presentation and all demo code is available on GitHub:

https://github.com/jvaneikeren/ChemicalStructureSearchingDemo