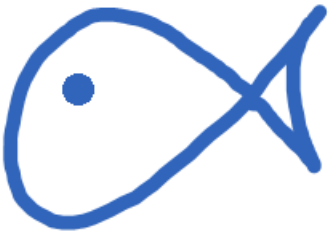
TrawlExpert: A Tool for Watershed Biological Research

Trawlstars Inc. (Group 11)
Lab section: L01
Version: 1.0
SFWRENG 2XB3

Christopher W. Schankula, 400026650, schankuc
Haley Glavina, 001412343, glavinhc
Winnie Liang, 400074498, liangw15
Ray Liu, 400055250, liuc40
Lawrence Chung, 400014482, chungl1

April 11, 2018



Revision History

Revision	Date	Author(s)	Description
1.0	05.04.18	HG	created
1.1	08.04.18	HG	algorithmic analysis added
1.2	10.04.18	RL	added Modular structure and uses
1.3	11.04.18	CS	added 2.3.x subsections
1.4	11.04.18	LC & CS	final revisions

By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application being developed through SE-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

Team Contributions

The individual contributions of each team member are described below. Subteam B indicates an algorithmic focus in a member's efforts while Subteam A indicates a focus on data parsing and user interface development. Although the contributions have been separated such that each task is recorded under one contributor, members often overlapped duties and designed modules together.

Name	**Role**	**Contributions**
Lawrence Chung	Head of Room Booking Subteam B Member	Implemented the depth first search, connected components algorithms, client code for connected components, Graph building, Node class and Bag class. Oversaw editing of Final Design Document.
Haley Glavina	Meeting Minutes Administrator Subteam B Member	Implemented the red-black tree, quickselect, and mergesort algorithms. Wrote `Histogram` function to sum each year's total individual count. Designed the final presentation slideshow, recorded and submitted all meeting minutes, and assembled the final design specification in LaTeX. Generated UML state machine diagrams.
Winnie Liang	Project Log Administrator Subteam A Member	Implemented the module responsible for parsing our data to create related objects (FileProcessor), implemented TaxonNode ADT. Led user interface development, set up tomcat files and directory structure, handled communication between the Google Maps APIs and JavaScript code. Overlooked and maintained project log entries.
Ray Liu	TA & Professor Liaison Subteam A Member	Implemented Record ADT, Date ADT, parsing API calls for WORMS API, RangeHelper for Basic Search, histogram output for command line, histogram output for web interface. Oversaw editing of Requirements Specification Document.
Christopher Schankula	Team Leader Subteam A Member	Determined the goals for each meeting, implemented the k-d tree algorithm, wrote backend of server, wrote command-line tool, implemented the BasicSearch and BasicSearchResult classes, implemented RecordCluster ADT, implemented TrawlExpert model class. Generated UML class diagram.

Abstract

TrawlExpert is a powerful tool to enable researchers to analyze and filter large datasets from fish trawl surveys in order to perform environmental research on fish and invertebrate populations. The tool gives researchers the ability to intelligently filter and query datasets based on biological classification such as family, genus or species, or based on location or timeframe. Advanced outputs display data as a histogram or geographical map, each depending on population abundance as a function of time and spatial distribution. Additionally, *TrawlExpert* provides a tool for finding local subpopulations within a larger query. A dataset of thousands of Great Lakes trawl surveys from 1958-2016 will be used as a demonstration of *TrawlExpert*'s capability to help researchers narrow down large datasets and glean data which pertains to their research. *TrawlExpert* will be designed to be used easily and effectively as the first step in a groundbreaking climate and ecological research pipeline.

Contents

1 Project Scope

1.1 Objective

Provide a statistical and visual tool for the analysis of water ecosystems, based on scientific water trawl data. Gives researchers with tools to analyze large datasets to find patterns in fish populations, including the plotting of historical population data on a map, the analysis of population trends over time and the determination of subpopulations of a certain biological classification.

1.2 Motivation

The diminishing of fish populations in the Great Lakes became a problem in the latter half of the 20th century, with the total prey fish biomass declining in Lakes Superior, Michigan, Huron and Ontario between 1978 and 2015 (Kinnunen, 2017). Annual bottom trawl surveys involve using specialized equipment to sweep an area and are used to determine the relative temporal variation in stock size, mortality and birth rates of different fish species (Walsh, 1997). These surveys are performed annually and often have hundreds of thousands of records, making manual analysis infeasible. The ongoing protection and development of the Great Lakes water basins is considered an important topic for scientists in both Canada and the United States, as evidenced by grants such as the *Michigan Sea Grant* (Michigan State University, 2018).

TrawlExpert will give researchers tools to filter through these large amounts of data by allowing them to search through data based on class, order, genus, family or species. This will help support scientific researchers and fishing companies as they study fish populations. These studies help inform initiatives to preserve fish populations and conduct their business in an environmentally friendly way going forward. As more data is collected on an annual basis, TrawlExpert can easily be injected with the new data and will adjust and scale accordingly, combining the new data with the old data for continued analysis.

TrawlExpert will also analyze the trawl data to find connected subpopulations within the data, giving researchers tools to analyze the portions of the water body that contain different populations and even track these specific subpopulations over time.

The focus of the project will be to develop these unique data searching and querying tools as a first step in a complete trawl survey analysis. For a complete analysis, tools like stratified statistical analysis are required by the researcher (Walsh, 1997). For purposes of maintaining a manageable scope for this project, the implementation of advanced trawl survey scientific and statistical analysis tools will be relegated to future developments.

1.3 Dataset

The test dataset that will be used for purposes of this project is the *USGS Great Lakes Science Center Research Vessel Catch Information System Trawl* published by the United States Geological Survey (United States Geological Survey, 2018). Compiled on yearly operations taking place from early spring to late fall from 1958 until 2016, the dataset contains over 283,000 trawl survey records in the five Great Lakes, including the latitude and longitude co-ordinates and biological classification such as family, genus and species.

1.4 Final Product

The final product included a command-line interface as well as a web interface using Apache tomcat. For information on launching the console, please refer to the `DEPLOYMENT.txt` document in the submission archive.

Apache tomcat was used to create a webserver which uses the internal functionality and model of *Trawl-Expert* written in Java. The UI allows users to filter by using information about different taxa (their biological relationships to each other, such as family / genus / species, etc) and display several different data outputs such as histograms, heatmaps, maps and population clusters, in addition to viewing raw data in tabular

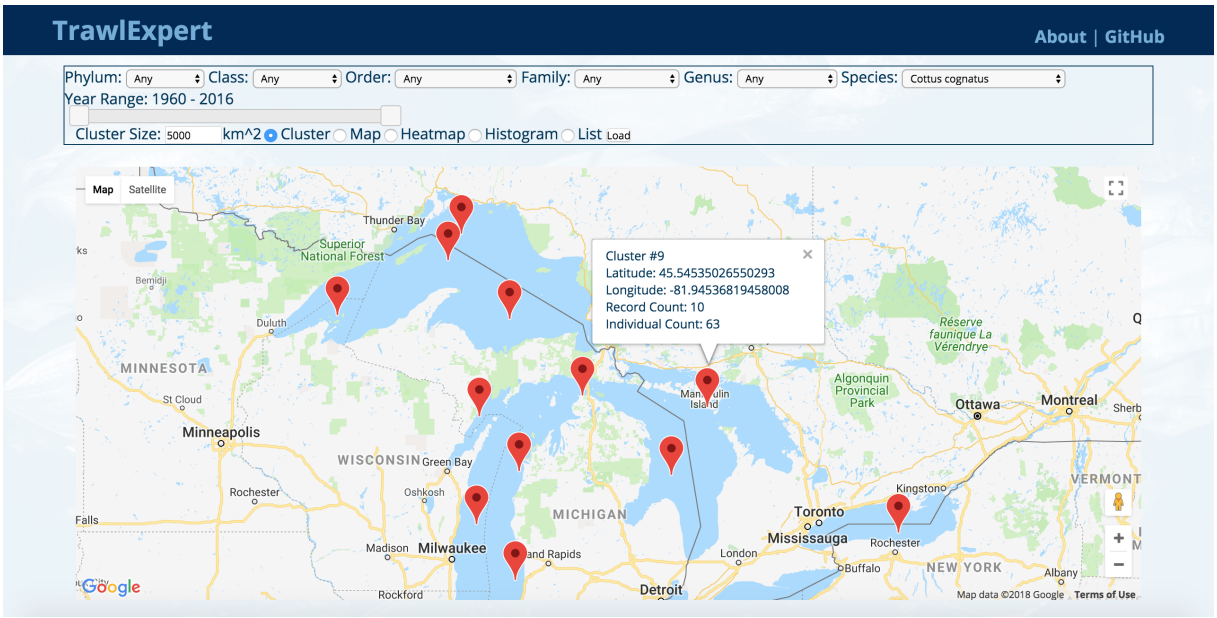


Figure 1: The final TrawlExpert web interface allows users to intelligently search for specific taxa and display several helpful statistical and data visualization tools such as maps and histograms. A live version of the web interface can be found at http://trawl.schankula.ca/Trawl

form. The clustering function is shown in figure 1. The *TrawlExpert* is hosted on Google Cloud Platform and can be accessed at http://trawl.schankula.ca/Trawl.

1.5 Glossary of Terms

Classification tree: Tree describing the relationships between taxa (for example, a species is the child of its genus).
Taxon (plural Taxa): Refers to a classification of a type of biological entity (species, family, etc)
Taxon ID: The ID given to a taxon in the WORMS database.
Trawl survey: Using Trawling to collect scientific data (Walsh, 1997).
Trawling: A method of fishing by dragging a net along the bottom of a body of water (Walsh, 1997).
WORMS: World Register of Marine Species, an openly accessible online database.

2 Implementation

The implementation involved over 30 classes implemented in Java. Additional JavaScript and HTML files were used to create a sophisticated web-based user interface. For a complete description of each class and module used, JavaDoc documentation can be viewed by opening the `doc/index.html` file in the submission archive or by visiting http://trawl.schankula.ca/Trawl/doc.

2.1 Modular Structure and Uses

The requirements in the Requirement Specification Document are achieved by designing modules with modularity and separation of concerns as a guiding principle. The relationship amongst modules is shown in the UML diagram in figure 2. Table 1 maps requirements to each class supporting that requirement. For a complete description of uses relation and public and private methods and modules, please refer to the JavaDoc documentation which can be viewed by opening the `doc/index.html` file in the submission archive or by visiting http://trawl.schankula.ca/Trawl/doc.

Table 1: Modules contributing to requirements
FR: functional requirement, **NF:** non-functional requirement

Requirements	Contributing modules
FR1. Reading input file and correcting corrupted data	WormsAPI.java, FileProcessor.java
FR2. Generate output file	*none — replaced by command-line tool and web GUI*
FR3. List of species by family, order, genus	BioTree.java, TaxonNode.java
FR4. Historical distribution of records	Histogram.java, RedBlackTree.Java, BasicSearch.java
FR5. Basic search by family, genus, species, time, location, etc.	BasicSearch.java, KDT.java
FR6. Geographical subgroupings	Cluster.java, RecordCluster.java, Graph.java, CC.java
FR7. Plotting/mapping tools	BasicSearch.java, CC.java, Google Maps API, tomcat)
NR1. Accuracy	KDT.java, TestBio.java, TestBasicSearch.java
NR2. Robustness	FileProcessor.java, BioTree.java, Main.java, tomcat server
NR3. Fast speed	KDT.java, QuickSelect.java, and RedBlackTree.java
NR4. Low memory usage	BioTree.java, DataStore.java
NR5. Scalability	KDT.java, QuickSelect.java, and RedBlackTree.java
NR6. User-friendly	Main.java, tomcat, Google Maps API, Graphical histogram

2.2 Class Organization

An overview of the modules included in the *TrawlExpert* is shown in figure 2. The *TrawlExpert* implementation efforts were divided into two subteams: Subteam A and Subteam B. Tasks were assigned so as to maximize the parallelization of the development process. As a biproduct, modules were designed to uphold the principles of information hiding, modularity and separation of concerns.

Using Java packages, related modules were grouped together to form packages of closely-related logic, as well as to ensure that modules were organized and easy to find. This section describes a package-level description of each one and justification for this modular decomposition.

By breaking up modules into generic pieces in this way, the team was able to parallelize development efforts and agreement on APIs allowed for smooth integration between the subteams' efforts. Effective communication ensured that while subteams were working on independent projects concurrently, all team members were aware of work being done throughout the team.

2.2.1 model Package

This package includes the class representing the model of the *TrawlExpert* platform. `TrawlExpert.java` provides wrapper hooks to function deeper inside the codebase, which allows different views to be used separately from the internal functionality. Two such views are the command-line interface in `main/Main.java` and the web interface through the tomcat server. This design allows any such view to be built upon the foundation of the model, without needing to edit any underlying code.

2.2.2 data Package

The top-level `data` package contains three main types of modules: abstract data types (e.g. `Record`, `TaxonNode`, `Date`), abstract objects (e.g. `BioTree`, `DataStore`) and supporting modules such as `WormsAPI`.

Abstract data types such as `Record` are classes for representing and accessing certain types of information in our dataset. For example, `Record` is the representation of a single line of the dataset (including taxon ID, latitude, longitude and date). Each ADT was designed to uphold the principle of encapsulation and provides methods for accessing data relevant to the ADT.

Abstract singleton objects were used to provide a central storage location for data. For example, BioTree contains information about the biological classification within the dataset. Since they are singleton objects, data can always be accessed from anywhere else in the program which needs it (see figure 2) without needing to pass references to objects. Similarly, DataStore provides a central location for the storage of Records in the dataset in a kd-tree structure.

2.2.3 search Package

The `search` package provides several generic search structures such as Red-Black Tree and kd-tree. It also contains a sub-package search.trawl which contains client code that utilizes these search structures to provide

TrawlExpert-specific searching.

Firstly, search and `search.kdt` provide generic kd-tree and Red-Black Tree implementations. These each utilize the `GeneralCompare` interface for maximum generality.

Secondly, `search.trawl` provides the `BasicSearch` module. This module is the access point for range-searching the dataset of `Record` objects in the `DataStore`. The `BasicSearchResult` class is the type of object returned from a range search, providing metadata about the search (such as the amount of time taken to find the results) as well as wrapper functions for generating histograms, clusters and sums of the data. In the case of histogram and sum, these results are cached after the first time it is called to improve performance if those fields are accessed again.

2.2.4 sort Package

The `sort` package includes modules such as `MergeSort` and `QuickSelect` as well as the custom interfaces `GeneralCompare`, `GeneralRange` and `Field`.

The `MergeSort` package was designed using generic typing and using the `GeneralCompare` interface means that it is even more flexible than using the regular Java `Comparable` interface. This class ended up being largely replaced by the `QuickSelect` class which served our purposes better. However, **MergeSort** is still used for sorting scientific names for the web interface, a small but integral part of the UI.

2.2.5 graph Package

The `graph` package provides functionality and abstract data types for constructing undirected graphs with a given number of nodes. It also contains the `CC` class for outputting connected components of the graph in question.

Additionally, the `graph` package includes the client code relevant to *TrawlExpert*, including `Cluster.java` and `RecordCluster.java`. These provide the functionality for clustering a given section of the dataset into related subpopulations based on a user-defined area of similarity amongst the records.

2.2.6 utils Package

The `utils` package contains miscellaneous helpful utilities for the *TrawlExpert*. At present, the only contained class is `Stopwatch.java` which provides the method for timing how long a `BasicSearch` query takes to complete.

2.2.7 web Package

The `web` package contains the controller (`Director.java`) and the startup scripts (`StartUpContext.java`) for the tomcat server. `Director` routes requests to the correct views in a model-view-controller-based setup. `StartUpContext` loads the TrawlExpert data into persistent memory on the tomcat server, allowing for fast lookups of data from memory.

Most of the other Java Server Pages (.jsp) files are contained in `tomcat/webapps/Trawl/`. Since this is not a focus of the project, these are not covered further in this design document.

2.2.8 test Package

The `test` package contains JUnit test cases for modules in the program which help to show that the various functionality is functioning as intended.

2.3 UML State Diagrams

Two UML state machine diagrams are included to describe the states and transitions within the *BioTree.java* and *Main.java* class.

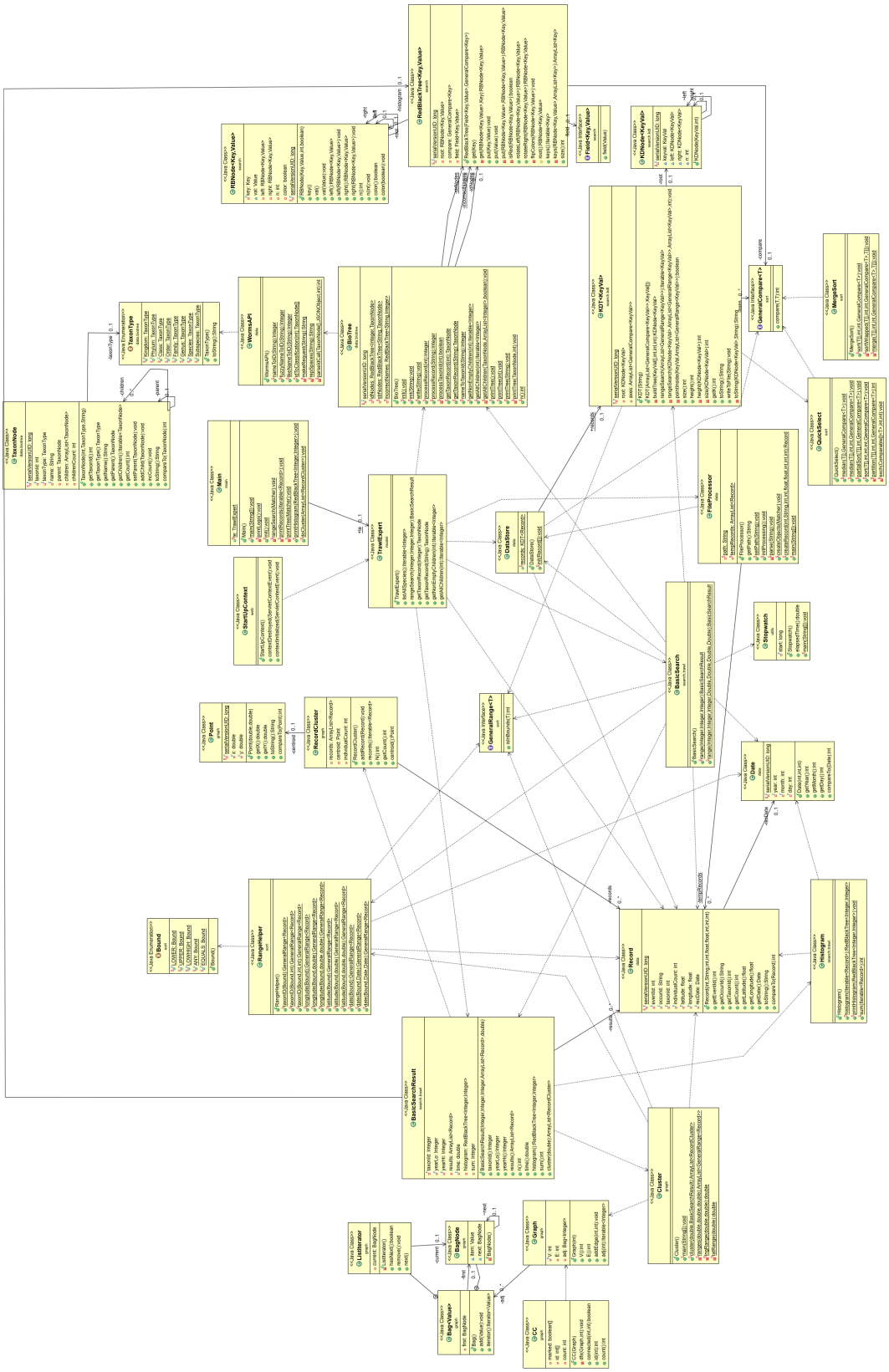


Figure 2: The UML class diagram of TrawlExpert, showing the relationships amongst modules. The full-resolution version is available in the root directory of the submission archive as `UML.png`.

2.3.1 Main.java

The UML diagram for the Main.java class is shown in figure 3. This represents the *TrawlExpert* console application's states, giving an overview of the types of queries and functions the user has access to. Since the *Main.java* class is a console version of the final server implementation, the states shown in its UML state machine diagram are analogous to many of the states of the final *TrawlExpert* website.

2.3.2 BioTree.java

The UML state diagram for the BioTree module is shown in figure 4. The BioTree class is a singleton class which stores the information about the different taxa in the dataset. This method has a few advantages. Firstly, the string names and relationships amongst taxa (e.g. species, genus, family) are stored only once and accessed when needed, saving large amounts of memory. After running through the *TrawlExpert*, the serialized dataset representing the same data is only 27mb, requiring very little storage on the user's computer.

Secondly, this diagrams represents a key feature of *TrawlExpert* in that it is able to recover corrupted data as the dataset is processed, which is very helpful for large datasets. In the USGS dataset, for example, there were 115 instances of different incorrectly named taxa, totalling 15,596 records (almost 6% of the records in the dataset). Using this method, these records were able to be recovered for proper use by the scientist. Using smart caching of incorrect names described by this UML diagram, the number of API calls to WORMS is kept at a minimum and the dataset processing only takes about 3 minutes. After the initial processing, the BioTree and records are stored as serialized Java objects to the disc, and can be reloaded in less than 10 seconds.

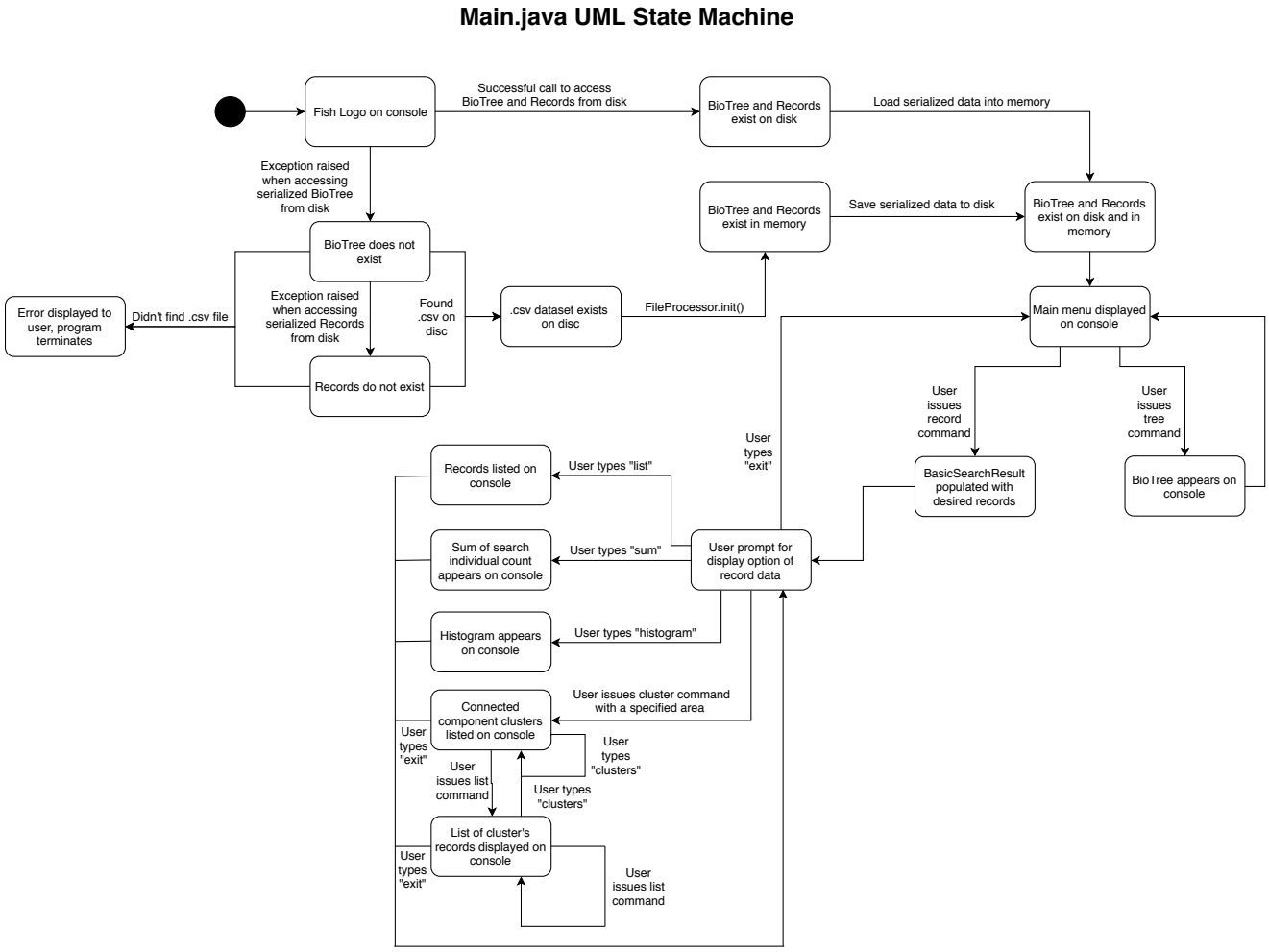


Figure 3: UML State machine diagram for *main/Main.java*, a class that provides console access to the *TrawlExpert*'s main functions. This class accepts search criteria from a user to produce a list of search results, depict a histogram of the records in that result, and compute a count of the search hits.

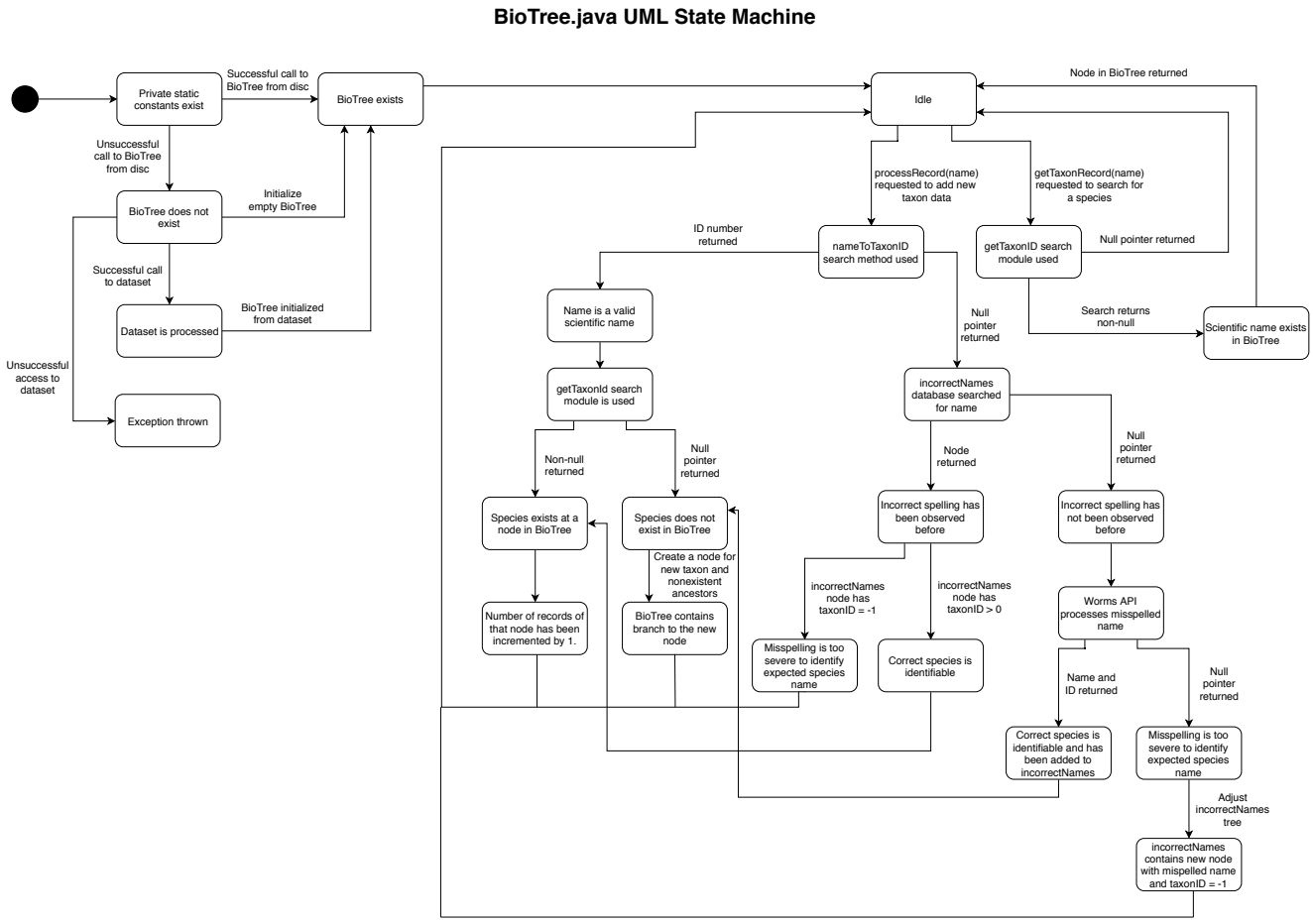


Figure 4: UML State machine diagram for */data/biotree/BioTree.java*, a class that builds a tree data structure from the scientific name hierarchies (taxa) of fish. Uses the World Register of Marine Species (WORMS) API to identify the correct spelling of species names for misspelled scientific names in the dataset.

3 Algorithmic Opportunities

The *TrawlExpert* was made possible by the use of vaious algorithms studied in *SFWRENG 2C03: Algorithms* offered at McMaster University. These algorithms include *Red-Black Tree* for searching and *Merge Sort* for sorting objects. Additional algorithms outside of the course scope were implemented to optimize the program; they are described below.

3.1 Quick Select

A modified form of the *QuickSort* algorithm that returns the k^{th} largest element of an unsorted array. Similar to *QuickSort*, *QuickSelect* randomly chooses a partitioning element to sort the array such that all elements smaller than the partition are left of it, and larger elements are to the right. However, rather than recursively sorting both halves of the partitioned array, *QuickSelect* only sorts the half containing the k^{th} index. The algorithm terminates once the partitioning element ends up at the k^{th} index of the array, the value of this element is returned.

This algorithm is implemented in */sort/QuickSelect.java*. It is used during the construction of *k-d trees* which require the frequent division of an array into two equally sized halves. By finding the median element of an array, it is partially sorted into equally sized small and large halves. The *QuickSelect* class implements a *median* method to simplify its usage in *k-d tree* construction.

3.2 kd Tree

A k-dimensional (kd) binary search tree was used to provide a fast range searching structure for the records. Given that the current size of the USGS dataset is over 280,000 entries and likely to grow with additional studies, it was crucial to have a structure to support fast searches. However, since the data contains many dimensions (taxon id, latitude, longitude, date), a simple binary search tree is not useful for this task. Instead, a kd-tree was employed.

A kd-tree is like a binary search tree except that on each level i of the tree, the comparison between nodes is made on the $(i \ \% \ d)$th axis of the data. For example, in a two-dimensional tree of (x, y) points, the first level of the tree would be compared on x, and the second level would be compared on y, the third level on x, and so on.

This structure gives a fast way of range searching for values in the tree, with a search complexity of $\mathcal{O}(dn^{1-\frac{1}{d}})$ where n is the number of nodes in the tree and d is the number of splitting dimensions of the tree. In the *TrawlExpert*, we use a 4-d tree to split on taxon id, date, latitude and longitude. Range searches for specific species are very fast, often on the order of 5-10ms and sometimes as fast as 1ms.

In order to build the kd-tree in a balanced way, it's crucial to be able to find the median of the data, so that a balanced number of nodes are inserted on each left and right subtree within the tree. In order to support fast *kd* tree building (which only needs to happen once when the dataset is first analyzed), the aforementioned *QuickSelect* algorithm was used, which was able to allow building the whole kd-tree in about 0.6 seconds. The kd-tree class then contains methods for serializing the data of the kd-tree so that it can be reloaded quickly from the disc on subsequent launches of *TrawlExpert*.

3.3 Graphing

Graph algorithms were used to support advanced searching features. Firstly, the biological classification of each organism forms a tree from which species in the same genus, for example, can be located. This was accomplished by creating a BioTree node, which stores the taxon id number of the classification, the scientific name of the entry, the number of records with that taxon id contained in the dataset and pointers to the parent and the children of the node. This structure directly mimics the method that scientists use to classify species according to their similarities (into family, genus, species) and allows for intelligent filtering and searching of the dataset. For example, with this structure it is possible to find all descendants of a certain biological classification.

Secondly, a graph algorithm was used to find connected components among search results. Nodes are connected together based on their distance to surrounding points (Tom10, 2012). Depth-first search was used to determine connected components (Broder et al., 2000).

4 Software Design Principles

4.1 Robustness

Robustness is a non-functional requirement prioritized during the *TrawlExpert*'s development. Considering all 280,000+ records in the dataset were entered by humans, data entry errors were inevitable. The *TrawlExpert* implementation had to ensure unexpected entries in the dataset were handled gracefully and could be recovered if possible.

When building a BioTree from the dataset, the World Register of Marine Species (WORMS) database API was used to find the correct scientific name of slightly misspelled names. Unless a name was severely misspelled, the Worms API was able to salvage small data entry errors. This ensured records could be used when building the BioTree and protected the tool from raising exceptions from small input errors. While this introduces a dependance on an Internet connection to *TrawlExpert*, it was assumed that the scientists working with *TrawlExpert* would have access to an Internet connection, and the tradeoff is reasonable for the recovery of many errors in the dataset.

The use of drop-down boxes on the user interface helped limit invalid search criteria from being entered. From left to right, each box contains increasingly specific components of a scientific name for fish species. When any of the dropdown boxes were selected, all boxes to the left (representing more general components of

that species name) were updated. This was to ensure the hierarchy formed by the more general components contained the newly adjusted value. Additionally, all boxes to the right were cleared. If a more general feature was adjusted, the resultant possible species no longer satisfies the hierarchy needed by values populating the right-most boxes. To prevent invalid scientific names from being used as search input, they had to be cleared.

4.2 Scalability

The tool must be able to handle large amounts of data, all while being able to complete queries at a high speed. Currently, the tool uses a dataset of 200,000 lines of data, but it must be able to maintain its high performace for larger datasets. Using sorting algorithms such as *Quick Select* to build a *k-d tree*, the *TrawlExpert* has been optimized to complete tree construction much faster.

Implementing *Quick Select* rather than *Merge Sort* drastically improved the *TrawlExpert*'s performance. When using *Merge Sort* during *k-d tree* construction, an array must be fully sorted before retrieving the median element, taking $\mathcal{O}(n \lg n)$ time where n is the size of the dataset. *Quick Select* only partially sorts the array before reaching the median, taking $\mathcal{O}(n)$ time, and it reduced *k-d tree* construction from 40.083 s using *Merge Sort* to 0.56 s, representing a 72x improvement.

4.3 Generality

A common theme among *TrawlExpert* classes is the use of lambda functions characterized by Java interfaces which describe their syntax as well as their semantic meaning. Lambda functions provide the capacity for parameterized object comparison or parameterized value access. This maintains the generality, and therefore reusability, of each class by allowing for generic types in class definitions. Type(s) of the input(s) and the how input object(s) are used only become assigned when the function is used.

4.3.1 General Compare

The *GeneralCompare* interface can be found at */sort/GeneralCompare.java*. This interface includes a *compare* function that takes two generically typed inputs and produces an integer output. When *GeneralCompare* is used in other classes, a compare function (the lambda function) is used to instantiate the expected input type and designate how the integer result must be calculated. This allows reuse of the interface among modules that perform comparisons of differently typed objects. Two records consisting of a fish species, date of observation, and geographic location can be compared based on lexicographic order of their names, date, or proximity to some location. *GeneralCompare* enables the comparison of record objects based on any of these parameters.

4.3.2 Field

The *Field* interface can be found at */search/Field.java*. This interface includes a *field* function that retrieves a key (a generic type) from a generically typed input object. Similar to *GeneralCompare*'s *compare* function, *field* is a lambda function. The field interface is used to perform searches in a tree of records that have been sorted by variable attributes from each record. The lambda function specifies which attribute to access when searching through the tree.

4.3.3 General Range

The *GeneralRange* interface can be found at */sort/GeneralRange.java*. This interface includes a *isInBounds* function returns an integer to describe if a record is member to a subset of the search results. The input has a generic type, rather than *Record* type, to satisfy reusability. The lambda function uses the range itself to perform conditional checks about whether the input object is below, within, or above the range. A return value of -1 indicates it is below, 0 indicates it is within, and 1 indicates it is above the range.

5 Internal Review

Table 1 shows the modules responsible for helping to meet the functional and non-functional requirements laid out in the Requirements Specification Document. This section describes in more detail how these were met.

5.1 Meeting Functional Requirements

The first challenge in developing this tool was parsing the data. One requirement was to read and clean the data, then produce a data structure of Record objects (**FR1**). The software tool performed this task as planned, and even exceeded expectations by using a *k-d tree* to store the Records in an easily and quickly accessible manner and the `BioTree` module had tools to recover corrupt data. **FR3** described a requirement to be able to list related taxa (family, genus, species, etc) in an intelligent way, which was achieved by the `BioTree` module as well. **FR4** was achieved by the `Histogram` module which generated a histogram that was displayed on the command-line and web GUI interfaces. Another requirement was to accomplish basic searching capabilities based on input criteria (**FR5**). This aspect was achieved through efficient sorting and searching algorithms, the results were verified using JUnit test cases of all searching and sorting algorithms. The geographical subgroupings (**FR6**) was achieved by using an undirected graph structure and connected component analysis.

The requirement of outputting records to .csv was not completed **FR2** as it was replaced by efforts in the command-line interface and web GUI. It should be explored further in the future.

5.2 Meeting Non-Functional Requirements

In terms of meeting non-functional requirements, the team met expectations. The use of the WORMS API when parsing the dataset improved robustness (**NR2**) and algorithmic choices such as *QuickSelect* and *kd-tree*s improved performance (**NR3, NR5**). The final product achieved the requirement of being user-friendly (**NR6**) since it is easily accessible via the Google Cloud server and prevents the user from entering invalid search criteria.

Additional goals included using less than 1GB of RAM (**NR4**), this was achieved since the *TrawlExpert* used approximately 0.5 GB of RAM. One way this was achieved was storing duplicate data (e.g. scientific name strings) only once. An additional goal was to perform queries in less than 1 second (**NR3**), this was achieved, with many queries taking approximately 1-10ms.

A positive team dynamic throughout the development process ensured collaboration and help were always offered, this was a large contributing factor to the success of the final product.

5.3 Changes During Development

There were some algorithmic changes that were realized during development. Two key algorithmic changes were the change from *Merge Sort* to using *Quick Select*, and changing cluster groups of Connected Components. As discussed in Algorithmic Opportunities, the use of *Quick Select* dramatically improved performance.

Another algorithmic change involved the client code for Connected Components when determining fish clusters. Initially, every node was visited multiple times to determine whether other nodes were within a given radius. The running time was unacceptable using this approach, and as a result, the algorithm was changed such that visited nodes were not revisited. This decreased running time significantly and was considered acceptable by the team.

5.4 Future Changes

While *TrawlExpert* met all of its original goals for this stage of its development, there are several points for improvement and future development of the platform as an all-in-one research tool for watershed research.

5.4.1 Improvements on Development Process

Most of the changes that would benefit the *TrawlExpert* involve its development requirements. The original goals for this section were quite extensive, however one aspect that was overlooked was file organization. Although GitLab was used for version control, confusion still occurred over which packages certain classes belonged to. For example, there were instances in the project where a search class would be located in the graph package. Adding a requirement for file organization would make the project more easily accessible in the development process and would also yield more efficient workflow because less time would be dedicated to searching for a desired class.

5.4.2 Future Functionality

Functionally, there are many future goals in the development of *TrawlExpert*. This phase of the development process was aimed at providing scientists with an effective tool to search and filter data relevant to their research, as well as some basic statistical tools. However, this only represents the first stage in a larger scientific research pipeline. Often, more advanced tools such as stratified statistical analysis is needed to properly take into account the many variables in trawl survey expeditions (Walsh, 1997). The future work includes building these tools into *TrawlExpert* in order to create an all-in-one research platform for trawl surveys.

Additionally, the original requirements specification stated that the program should be able to output subsets of the dataset to .csv files (**FR2**). This was replaced by the command-line interface in `Main.java` and the web GUI, but should be explored further in the future as it would be helpful to scientists interested in obtaining raw data for further analysis.

References

Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tomkins, A., and Wiener, J. (2000). Graph structure in the web. *Computer networks*, 33(1-6):309–320.

Kinnunen, R. (2017). Great Lakes prey fish populations declining. http://msue.anr.msu.edu/news/great_lakes_prey_fish_populations_declining_msg17_kinnunen17.

Michigan State University, U. (2018). Michigan sea grant. http://www.miseagrant.umich.edu/.

Tom10 (2012). 2d point clustering. https://stackoverflow.com/questions/3937663/2d-point-clustering.

United States Geological Survey (2018). USGS Great Lakes Science Center Research Vessel Catch Information System Trawl. https://www1.usgs.gov/obis-usa/ipt/resource?r=usgs_glsc_rvcat_trawl.

Walsh, S. J. (1997). Efficiency of bottom sampling trawls in deriving survey abundance indices. *Oceanographic Literature Review*, 7(44):748.