MEng Group Project Specification & Design

Cristian Badoi, Aaron Butterworth, Daniel Gardam

1 Project Summary

The aim of this project is to create a system capable of calculating, storing and predicting formulae of chemical compounds given a set of precursors, with a focus on optimization and finding an acceptable solution within a given set of bounds. The program is designed to be remotely accessible, with a web server providing a user front end and a space for necessary calculations, and a database for the purposes of storing chemical data and user account information.

The project will primarily use C++ for computation, NodeJS with several popular packages for the web server and user interface, and MongoDB for the database system.

This project is produced in conjunction with the Materials Innovation Factory and is intended for their use. The development and evaluation of the project will be performed in cooperation with the facility.

This project is mainly based on the prior research topics and problems encountered by researchers in the MIF, for example having a person test possible compounds and slowly refine the formula based on their results is a time and material inefficient process, therefore if we can suggest compounds which are either direct results or very close approximations to their desired outcome then this process can be minimised.

2 Design Overview

2.1 Expected Components

2.1.1 Web Server

The main interface to the system will be supplied by a web server to allow easy and portable access by members of the MIF. The web facing part of the system will be comprised of several smaller packages all working together under NodeJS:

NodeJS — A JavaScript runtime that allows execution of JS outside the context of a web browser.

Express — An open source web application framework for NodeJS, it is responsible for the high level web server logic and allows a high level interface to web traffic as it handles response codes, etc. itself.

Pug — An open source high-performance template engine, it allows us to alter the contents of a web page just before serving it to the user, allowing for finer levels of customisation while keeping the code base manageable.

Express-Session — An add-on for express that gives us easy control of user sessions. This allows us to keep track of user specific data, such as preferred precursors or recently searched compounds.

bCrypt2 — A widely trusted package for providing user password encryption.

2.1.2 Computation

The actual computation will be handled by C++ programs with some small python helper scripts for unit conversion and other pre/post processing.

This is done for several reasons, firstly to allow a significant performance increase over JS; secondly to allow us access to widely known mathematics libraries; and thirdly to allow easier modification of the source code by others in the MIF as these are their preferred languages as opposed to JS, this is also the start of an ongoing project which will be maintained and expanded upon after us so maintainability is an important factor.

The first step of computation is the stoichiometry calculator. This takes a selection of user defined elements, and then calculate all possible balanced compounds that may be produced by a chemical reaction of these elements. This process also takes into account user limits such as the maximum amount of atoms in the resulting combination to refine the search and prevent the creation of a technically correct but practically unreasonable result.

The programs data sources are the web interface which will invoke the users query and the database for relevant information about each element, its output is piped back to NodeJS where it may be cached in the database if it is a common search, the result is then displayed to the user via the web interface.

The second stage is the precursor calculator. Its input is a desired ratio of elements, and a set of available chemical precursors. It calculates possible combinations and quantities of the precursors that when mixed together create the desired target ratio. User bounds may also be given to keep the results practically usable.

2.1.3 Database

With the theme of future expansion the database will be hosted on a separate server instance running independently of the NodeJS process, this allows for easier expansion of the system if it cannot handle the user load placed on it as both the web server and database can be scaled completely separately from each other without modification to the program code.

The database will not only store the information needed to run calculations, but will also store the solutions to commonly executed combinations so as to reduce the computational demand by allowing these solutions to be quickly retrieved rather than repeatedly calculated.

3 Algorithm Design

3.1 Stoichiometry Calculator

The queried elements are transformed into a matrix with the following properties:

If the user selected Al and O with a limit of 5 atoms in the result:

$$\mathbf{A} = \begin{matrix} \text{Al} & \text{O} \\ \begin{bmatrix} 1 & 1 \\ 3 & -2 \end{bmatrix} & \begin{matrix} \text{Initial Quantity} \\ \text{Charge Imbalance} \end{matrix} \end{matrix}$$

$$\mathbf{B} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{matrix} \text{Resulting Proportion} \\ \text{Desired Charge Imbalance} \end{matrix}$$

$$\begin{bmatrix} 1 & 1 \\ 3 & -2 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

This gives us the matrix form of the system of equations we must solve. However the charge imbalance in the matrix $\mathbf{A}$ is not constant for all elements, in this example O can exists with $-2, -1, +1, +2$ as its charge, solving the equations above only allows for Al_2O_3. Therefore we must also consider all permutations of element charges, in this example O has a total of 4 states, and Al a total of 3 giving a total of 132 permutations.

However, each permutation is quick to solve. Continuing the above example:

Via LU decomposition:

$$\begin{bmatrix} 1 & 1 \\ 3 & -2 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 1 \\ 0 & -5 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 \\ 3 & 1 \end{bmatrix} \times \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 0 & -5 \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0.4 \\ 0.6 \end{bmatrix}$$

While the example seems trivial when many elements are selected the system of equations grows rapidly and the merits of this method become more apparent.

This computation is run on all permutations of element charges, results are then collated, filtered, and then passed back to the web server where it can be presented to the user.

Caching can be done at this phase as some searches will be significantly more common than others depending on the precursors or particular interests of the user in question.

3.2 Precursor Calculator

The precursor calculator operates on almost the same principles as the stoichiometry calculator, however the means of constructing the matrices is different:

If the user selected Li_2S, Al_2S_3, Al_2O_3, $LiAlO_2$, Li_2O as precursors, and a desired end ratio of $Li_1 : Al_1 : S_1 : O_1$ we construct:

$$\mathbf{A} = \begin{matrix} Li_2S & Al_2S_3 & Al_2O_3 & LiAlO_2 & Li_2O \\ \begin{bmatrix} 2 & 0 & 0 & 1 & 2 \\ 0 & 2 & 2 & 1 & 0 \\ 1 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 \end{bmatrix} & \begin{matrix} Li \text{ in compound} \\ Al\text{'} \\ S\text{'} \\ O\text{'} \end{matrix} \end{matrix}$$

$$\mathbf{B} = \begin{matrix} \text{Ratio in output} \\ \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} & \begin{matrix} \text{Li} \\ \text{Al} \\ \text{S} \\ \text{O} \end{matrix} \end{matrix}$$

We then find the null space of $\mathbf{A}$, firstly we apply Gauss-Jordan elimination to put it into reduced row echelon form:

$$\text{rref}(\mathbf{A}) = \begin{bmatrix} 1 & 0 & 0 & \frac{1}{2} & 1 \\ 0 & 1 & 0 & -\frac{1}{6} & -\frac{1}{3} \\ 0 & 0 & 1 & \frac{2}{3} & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Taking the 'free' section of the rref($\mathbf{A}$) and multiplying by -1:

$$\begin{bmatrix} -\frac{1}{2} & -1 \\ \frac{1}{6} & \frac{1}{3} \\ -\frac{2}{3} & -\frac{1}{3} \end{bmatrix}$$

Then adding identity to these rows so the height of our new matrix is the same as the width of the initial matrix $\mathbf{A}$:

$$\begin{bmatrix} -\frac{1}{2} & -1 \\ \frac{1}{6} & \frac{1}{3} \\ -\frac{2}{3} & -\frac{1}{3} \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The null space is then defined by scalar multiples of the columns of this matrix:

$$x \cdot \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{6} \\ -\frac{2}{3} \\ 1 \\ 0 \end{bmatrix} + y \cdot \begin{bmatrix} -1 \\ \frac{1}{3} \\ -\frac{1}{3} \\ 0 \\ 1 \end{bmatrix}$$

Where x and y are any real numbers.

Finally, via the same method demonstrated in 3.1, we find a solution $\mathbf{S}$ to our initial system of equations:

$$\begin{bmatrix} 2 & 0 & 0 & 1 & 2 \\ 0 & 2 & 2 & 1 & 0 \\ 1 & 3 & 0 & 0 & 0 \\ 0 & 0 & 3 & 2 & 1 \end{bmatrix} \times \mathbf{S} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \qquad \mathbf{S} = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{6} \\ \frac{1}{3} \\ 0 \\ 0 \end{bmatrix}$$

Using this solution and the null space we can then find the space of all possible solutions to the system of equations:

$$\mathbf{S} + x \cdot \begin{bmatrix} -\frac{1}{2} \\ \frac{1}{6} \\ -\frac{2}{3} \\ 1 \\ 0 \end{bmatrix} + y \cdot \begin{bmatrix} -1 \\ \frac{1}{3} \\ -\frac{1}{3} \\ 0 \\ 1 \end{bmatrix}$$

Each solution we return is cached in the database as a possible point and can be presented to the user graphically via the web server.

4 Data Structures

The main source of data for the system is the MongoDB database. We are using it to cache results allowing us to lighten the computational load of the system and deliver results faster when several users query the system at once.

4.1 Database Structure

Database tables are generated on-the-fly as required, however general types of table do exist:

UserData — Usernames, Encrypted Passwords, and other user preference data.

ElementData — Charge configurations, and other static element data needed for calculations.

CalcedPoints — Generally a table per invocation of the precursor calculator, storing the results of our calculations so they can be easily referenced in future.

The exact structure of a CalcedPoints table depends heavily on the query that created it, however given a query its structure will always be predictable:

<Desired Ratio>		
<Precursor 1>	<Precursor 2>	...
<Ratio of P1>	<Ratio of P2>	...
⋮	⋮	⋱

Each row of the table is a possible solution with the values in each column representing the proportion of that precursor relative to the others.

4.2 System Memory Management

On the computer which runs the main web server instance availability of RAM may become an issue, this would mainly be caused by multiple large queries executing simultaneously and such a situation is hard to avoid, however the memory footprint of the application is kept to a minimum by invoking the calculators as child processes instead of keeping them with the main NodeJS thread.

This allows the memory heavy segment of the system to only take up space in RAM when it is required, and also only occupy the exact amount it requires during computation. If resources are not available for a query to use it will be queued until they are available on a first-come first-served basis.

4.3 Computation

The primary data structure in use during the computational stages are matrices, they are used to store the coefficients that represent the system of equations we have generated and will then solve. The software does occasionally cache small amounts of data for its own use, this is not stored in the database and is instead written and read as comma separated values in text files.