Implementation of the Artificial-Centered Hit-and-Run Sampling Algorithm in OptFlux

Bruna Daniela Azevedo da Silva[1]

MSc Bioinformatics, Escola de Engenharia, Universidade do Minho
ni.azevedo.s@gmail.com

Abstract. Flux Balance Analysis is frequently applied to metabolic networks models to obtain the maximum value of a specific reaction under a set of constraints. However, this method only provides an optimal solution from the boundaries of the solution space. The sampling algorithm generates points that cover the entire solution space, in which the result considers the entire flux distributions. Artificially-centered hit-and-run sampling (ACHR sampling) applies the hit-and-run sampling by giving it a center point in order to guarantee that the points stay within the solution space. Here we present, an efficient java implementation of the artificially-centered hit-and-run sampling algorithm. The algorithm is presented by an intuitive graphical user interface developed as plugin for the OptFlux workbench.

Keywords: Systems Biology, Sampling, OptFlux

1 Introduction

1.1 Constraint-based models

The reconstruction of *in silico* models of large metabolic networks are used to understand the complex metabolism of the cells.

A metabolic network is a graph represented by the connection of genes to the reactions catalyzed by the enzymes the genes codify, as well the consumed and produced metabolites(Figure 1a). To construct a metabolic network genome annotations, metabolic databases and published literature is needed. From the genome annotations we can relate the genes to the enzymes they translate and from the metabolic databases the connection between reactions and metabolites is established. The published literature will help with the kinetic data, energy costs as well as the stoichiometry of the reactions [1, 2].

This metabolic network can mathematically represent its information in a system, where each linear equation represents a metabolite. As the variation of metabolite concentration is maintained throughout the time, the equation Sv=0 is used in the system to construct the S matrix accordingly to its values and stoichiometry (Figure 1) [3]. In case the system is undetermined, there will not be a single solution, but instead, a set of infinite solutions (Figure 2a).

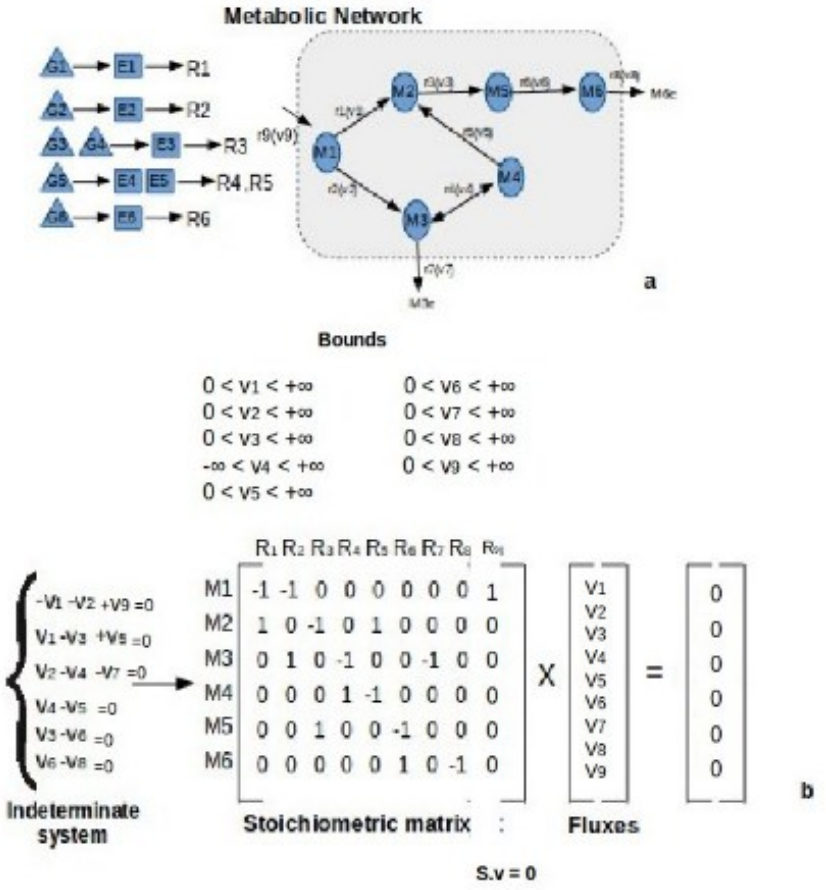

Fig. 1. Diagram of the construction of a metabolic model. There is a sample of a metabolic network containing 6 genes represented by a triangle, 6 enzymes represented by the rectangle, and 6 metabolites that are the circles, as well as 8 reactions (a). From the metabolic reaction, 6 linear equations are added to the system and the corresponding stoichiometric matrix and fluxes are obtained (b). Each reaction bounds are later added accordingly to its thermodynamic directionality.

Once the model is defined, a solution space is created containing all the possible solutions (Figure 2d) [4, 5]. The constraints applied in this tasks can be divided in three categories, compartmentalization and mass conservation(1), that defines the initial solution space, thermodynamic directionality(2,3) and the flux capability(4) [6]. The application of such methods is advantageous in genome-scaled models analysis since it involves a massive number of reactions and lacks kinetic data [7].

$$S.v = 0 \tag{1}$$

$$v_i \geq 0, \forall i \in N_{irrev} \tag{2}$$

$$-\infty \leq v_i \leq +\infty, \forall i \in N_{rev} \tag{3}$$

$$v_{imin} \leq v_i \leq v_{imax} \tag{4}$$

The first constraint dictates that the model is at steady-state.

The equation is represented by a stoichiometric S matrix of size mxn. m represents the number of metabolites and n represents the reactions of the model. Each value of the matrix indicates whether the metabolite is present in the reaction and its stoichiometry. If the value is positive the metabolite is produced in the reaction and if the value is negative, the metabolite is being consumed [3]. The vector v is an vector of fluxes [7]. Afterwards, a second constraints are set to the reactions obtained previously, which will represent the directionality. Then, a third constraint is applied to the bounds, the flux capability, yielding them a value. Most reactions are considered irreversible, but certain reactions of uptake or secretion permit the use of experimental values. The bounds can be placed based on the knowledge of the cellular thermodynamics or, in case it is not known, large entry values as 10000 can be set instead [2, 8].

Flux-Balance Analysis There are several methods for analysis of constraint-based models. One example is the Flux-Balance Analysis (FBA) which is a phenotype prediction method that uses linear programming to optimize an objective function by maximizing or minimizing parameters, such as biomass or metabolite production. In this process the objective it is to add another delimiter to the solution space, since it will fix a value to the flux that will be optimized, shortening the set of possible solutions. The solution will be then found in one of the vertices of the newly obtained space [2].

Although this method is extremely practical, it has several limitations. The dependence of the objective function whose influence could change from species to species. The results obtained can also be considered a limitation, as the analysis only returns a single optimal solution, so it cannot be applied to problems that have several objective functions. Besides, the solution will lie in the solution space boundaries, so the internal fluxes are not analyzed. Since the system is forced to have a single solution, it will not resolve all the possible solutions presented in genome-scale models and therefore being discarded [9, 8].

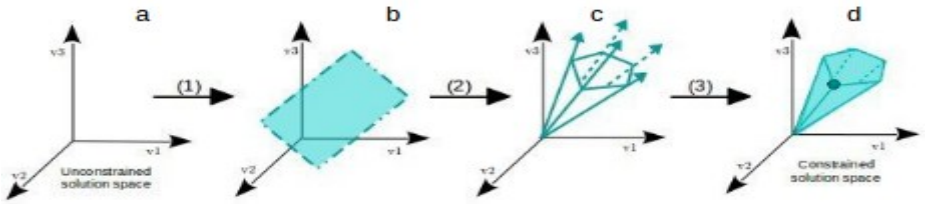

Fig. 2. Schematization of the application of the constraints on a model step-by-step. The numbers represent the equations shown above. In the first step, the model is not submitted to any constraint so it has an infinite set of possible solutions (a). As the first constraint is applied, all linear equations of the system will be equal to zero, establishing a solution plane (b), that later will take form by applying the second and third constraint, based on the direction of the reactions(c) and the maximum and minimal value for each of them(d). Finally a solution space with a finite set of solutions is obtained. The dot is an example of a solution given when FBA is used. Adapted from (Orth et al.,2010) [1].

1.2 Sampling

The sampling method takes in consideration the entire solution space returning a set of points that represent the flux distributions, covering all the convex space. Markov-chain Monte Carlo(MCMC), an algorithm known as "hit-and-run", is the main sampling algorithm in which Markov-chain represents a stochastic process of a sequence of random variables and the next event will only depend of the current state of the system[2].

The hit-and-run algorithm starts at a random point, moving to a point in random length and in random direction. The new point will be considered the new stating point and the same process will occur [10].

Following this process, the ACHR only modification is by using the sampled points previously obtained to calculated the boundaries and direction of the space [11]. This function addition was introduced due to a lack of assurance that the hit-and-run algorithm would not leave the bounded space and the enormous number of steps for it to cover the entire solution space using a uniform distribution [9].

The ACHR algorithm starts by creating the warmup points, where for each reaction a set number of random points are created in a random order. Then, a center point is selected based on the mean of the warmup points. The center point will be the initial point of this method. From the center point, a random direction, that can be either positive or negative, and a random distance are obtained. The distance is a random number from a calculated interval given after measuring the distance from the center point to the boundaries. Lastly, the next point will be the current point plus the distance multiplied by the

direction. After the reallocation from on point to another, the current point will be the center point and the process will continue until it reaches the desired number of iterations [12].

Although the iterations of the ACHR algorithm are not a Markov chain, because it depends on direction, so a uniform distribution is not guaranteed, this method is recommended when the solution space of our model shows irregular shapes. In this cases the use of an artificial center instead of a uniform direction renders a better option [2]. ACHR algorithm permits the use of a biased uniform distribution if there is a necessity to confirm that the results are in uniform distribution and biased normal distribution [12].

Throughout the years, this algorithm was improved, arising to different Sampling methods, the gpSampler and optGpSampler (combination of both methods). In case of the gpSampler, the sampling assumes by default a normal data distribution and the solution space is irregularly shaped, so the procedure to create the warmup points will be divided in two phases. The first phase is based on a linear programming that consecutively maximize and minimize every reaction in order to obtain the warmup points. In the second phase, the optimization is ruled by the random weights assigned to the fluxes [7]. On the other hand, while the OptGpSampler employs the same method to create the warmup points, similarly to the gpSampler, cutting the second phase as the weight assigned to the fluxes result in similar optimal solutions. The sampling procedure is similar to the ACHR sampling, however it only selects samples at k interactions and generates chains in parallel [7].

Currently, in terms of software, this algorithm is only implemented in the COBRA Toolbox for its MATLAB extension. Though this toolbox offers many implementations, just like the ACHR sampling, ACHR parallel sampling and gpSampler, it needs a paid license to use the MATLAB software. Furthermore, it requires basic knowledge of MATLAB language and the graphical results need to be later created. Our plugin was developed from the ACHR Sampling, whose procedure was described before, in MATLAB with a user friendly graphical interface and it can be combined with other simulations and analysis form the OptFlux software.

1.3 Frameworks

COBRA Toolbox The COBRA Toolbox is a MATLAB package for implementing constraint-based methods to genome-based models, making it possible to predict metabolic phenotypes. The result will be represented by a small set of solutions, giving the user hypothesis for the biological problem being analyzed. This package offers a vast number of methods, such as FBA, gene essentiality analysis, minimization of metabolic adjustment analysis, geometric FBA, creation of context-specific subnetwork models using omic data, Monte Carlo Sampling, including the ACHR and gpSampler, fluxomics and gap filling [12].

OptFlux OptFlux is a user-friendly, free software for metabolic engineering. It is capable of performing several analyses based on strain optimization, being

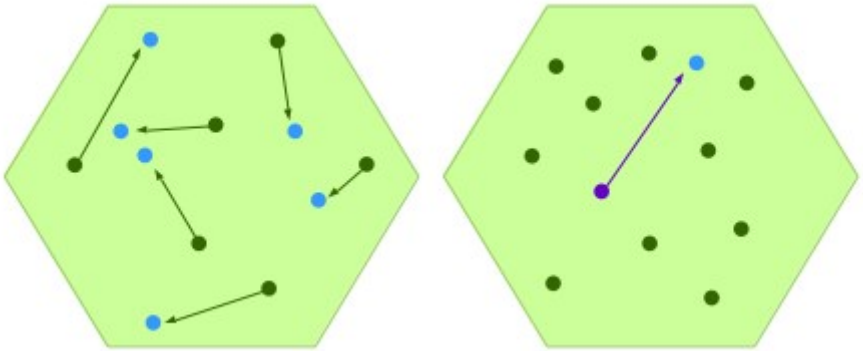

Fig. 3. Schematization of both MCMC sampling and ACHR sampling, from the left respectably. The MCMC sampling selects random points, represented by the green dots, chooses a random direction and a random distance, the green arrows, creating a new point, the blue dots. The ACHR sampling selects a set of random points named warmup points, the green dots, calculates the mean of the dots, a center point represented by the purple dot, and from there a random direction and distance is obtained, creating a new point, the blue dot. Both processes occur until all internal fluxes are analyzed.

the first to incorporate evolutionary algorithms and simulated annealing. This software is modular due to being built on top of a framework called AIBench. The user interface is constructed in GUI (Graphical User Interfaces).

The AIBench framework is an Artificial Intelligence workbench that utilizes data mining, a computational process of data patters analysis, extracting information from sets of data and rearrange it into understandable information [13]. It is programmed in Java programming language and provides a plugin architecture to the software permitting the addition of more modules. Its application follows the MVC (Model-View-Controller) design, used for the implementation of GUI. The model handles the logic of the data, the view displays the data and the controller handles the user interaction. The model stores the data and is manipulated by the controller, updating the view, so every time a user choses a new option, the view is updated. In this case, the controller will send the INPUT to the model and the view will display the OUTPUT of the model.

2 Implementation

2.1 Sampling

We developed an user friendly sampling algorithm divided in two packages, abstraction and implementation. The abstraction package consists in four classes, *Bias*, *CreateWarmupPoints*, *Sampling* and *SamplingResult* and an interface *ISampling*. Moreover, the implementation has the *ACHRSampling* class. This project is a reimplementation of the ACHR sampling from the MATLAB software to

Java. A description of the different classes that comprise our Sampling Algorithm as well as, all the alterations performed to accomplish a stable and properly working system is written bellow. *CreateWarmupPoints* creates a matrix of dimension consisting in number of reactions and number of points chosen, employing the model being tested, the chosen number of points and the bias, as its input. Using the *Bias* option, the user has the opportunity to insert preferences in the distribution of the flux values.

As attributes, the class has the method applied, normal distribution or uniform distribution, the index of the fluxes and its correspondent parameters, whose type can change depending on the method applied. In case of the normal distribution the parameters are the mu (mean) and sigma (standard deviation) and for the uniform distribution its the minimum and maximum value. This parameters will alter the lower and upper bounds of the reactions. For the uniform distribution, the parameters are used to calculate a number using the following equation,$(max - min) * randomnumber + min$. A multiplication of the calculated value with 0.99 is set as the lower bound and to the upper bound the multiplied value is 1.001. In the case of the normal distribution, the procedure is the same but the calculated value uses the equation, $sigma * randomnumber + mu$. There are important checkpoint methods present in the *CreateWarmupPoints* class, such as:

- *verifyPoints* - verifies if the number of chosen points is at least two times the number of reactions of the model.
- *methodUniform* and *methodNormal* - applies parameters to the bounds of the reactions.
- *meanWarmup* - calculates the mean of each row of the warmup points (centerpoint).
- *run* - creates the warmup points matrix.

In order to create the warmup points, it is necessary to, firstly, allocate the parameters of the bias option to insure that the values are between the model bounds and, secondly, create the matrix where the lines represent the number of reactions and the column is the chosen number of points. The Bias option, if activated, will change the bounds as described before, initializing the creation of the matrix. For each reaction the maximization and minimization is performed, within which the result of the optimization is one column the warmup points matrix. After the individual minimum and maximum of each reaction is calculated, a random objective function will continue to calculate the points, until the matrix is completed. Therefore, the matrix obtained will have the double amount of reactions of the model, because the optimization for each reaction is compulsory. In case the value is too low, the *verifyPoints* method will set to the minimum value allowed in the matrix, which afterwards, the warmup points can be recalculated by the addition of the centerpoint and several multiplications. The ISampling interface, which represents the behavior of the algorithm, only contains the activation of the *SamplingResult*. The *SamplingResult* is based on the calculated matrix where each line is a sampling point with a number of columns equivalent to the number of reactions.

In this algorithm the *Sampling* is an abstract class that will enclose all the implementations necessary for the correct analysis during the *ACHRSampling*, hidden from the user, extending only its functionality. Our abstract class contains the following methods:

- *meanRowMatrix* - calculates the mean of each row of the warmup points (centerpoint).
- *getDists* - returns a Map with the distance from the current point to the upper and lower bounds for each reaction.
- *closeBoundary* - verifies if the point is too close to a boundary. It is necessary to find the direction in each the point will be projected.
- *findDirection* - returns an Array with the direction. For the positive direction the integer option will be 1 and 0 for negative direction.
- *maxminStep* - calculates the maximum and minimum step size.
- *StepVector* and *addArray* - from the previous method, four step vectors are created, two for the maximum and two for the minimum, where each value is divided in positive or negative. The maximum positive and the minimum negative will be added together using an *addArray* function resulting in the maxStepVec. For the remaining vector similar operations are performed resulting in the minStepVec.
- *minmaxValue* - returns the true maximum and minimum step sizes. After the two doubles are obtained an if will decided if it is needed to find another direction because the point is too close to a constraint. If another direction is found the analysis will return to the beginning of the cycle.
- *reprojectPoint* - a new group of points will advance to the next point from the current point using a step distance. For example, the initial current point is the new previous point and the new group of points are the new current point.
- *findOver* - an auxiliary method to re-project the current point to the next step.
- *alterCurPoint* - an auxiliary method to fix the new points as the current point.
- *recalculaCentro* - the centerpoint is recalculated for the next iteration in the cycle.

The *ACHRSampling* extends the *Sampling* abstract class and its methods are implemented in one *for* and two *while* cycles, one *for* for the number of files and each *while* for the number of points and the steps per point that are performed. The initialization of the *ACHRSampling* is defined by the model obtained containing the information from the warmup points matrix obtained in the *CreateWarmupPoints* class, the number of files, the number of points per file and the number of steps per point. As referred previously, the final result of this algorithm is an *ArrayList* of *Map*, and so no file will be created, as the name only comes from its implementation in MATLAB. Both processes *ACHRSampling* and *Sampling* experienced several alteration to their code. The original template presented several incongruities that had to be redesigned or erased,

such as, presence of strings associated with the data analysis whereas, the aim is to employ manipulation of integers and doubles to obtain the overall result. Otherwise, only small mistakes in the saving of the points to the *SamplingResult* were found and therefore corrected. However, the Sampling class was the main focus of the work done and the one that suffered major structural alterations through deletion of methods and addition of more straightforward modes to perform this tasks. With this in mind, the following methods were created or altered: *findDirection*, *closeBoundary*, *maxminStep*, *findOver*, *addArray*, *StepVector*, *recalculaCentro* and *convertPointsToMap* and, afterwards converted from *Map* to *ArrayLists*, since the name of the reaction would not be necessary and reducing memory consumption.

A class diagram Schematization of this package is in appendix A, (Figure 8).

2.2 OptFLux Plugin

The plugin is an user interface implemented based on the Model-View-Controler architectural pattern. We designed a plugin for OptFlux that combines six packages *DataType*, *GUI*, *Operations*, *LifeCycle*, *Serializers* and *Views*. Since the plugin is constructed on top of AIBench, three components are usually represented in the Controler, which is the Operations, DataType that correlates to the Model and the Views. The *GUI* package employed in our project is a View for the Operations containing two classes the *SamplingOptions* and the *SamplingGUI*. This classes will create a graphical user interface to simplify the selection of the project and several parameters to perform the sampling (Figure 10). To choose the Bias, there are two tables, one that allows a faster method to find the desired reaction and a second, a Bias table where the reaction and its values will be added (Figure 11).

The events triggered in the GUI are described below:

- *updateComponents* (contains *calculate*) - calculates the total number of points that the ACHR Sampling will create. It multiplies the number of files, number of points and number of steps per point.
- *tableselection* (contains *updateReaction*) - inserts the reaction ID in the set bias area and indicates which reaction the Bias values will be implemented on, and therefore, added to the Bias Table referred previously.
- *selectfromCombo* (contains *chooseBias* and *changeHeader*) - updates the GUI in compliance to bias selection. The *chooseBias* method allows the addition of reactions to the bias table, enabling the disabled options and the "Set" button. Depending on the chosen bias, the values labels and the bias table headers change accordingly. *changeHeader* updates the table headers as the bias method changes.
- *setinformation* (contains updateInformation) - adds a new row to the bias table. This row is composed by the reaction ID, reaction name, the min/mu and the max/sigma values.
- *setRemoveButton* - enables the remove button if a row is selected in the bias table.

– *removeFromTable* - removes the selected row if the Remove button is clicked on.

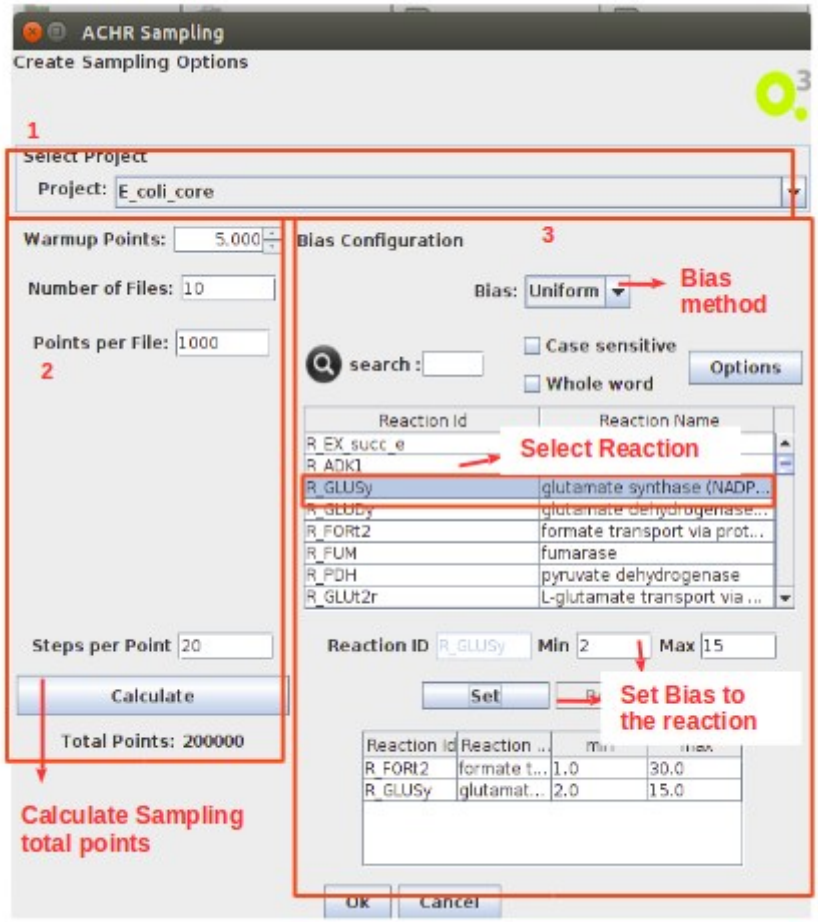

Fig. 4. Option Selection for sampling with bias option.

Afterwards, the parameters are received by *SamplingOperation* in a list format. This parameters are then used to perform the sampling algorithm.

The sampling results are retrieved and stored in the model *SamplingDataType* to be subsequently used in the construction of the final view. As mentioned, the last class necessary for the creation of the plugin is denominated View.

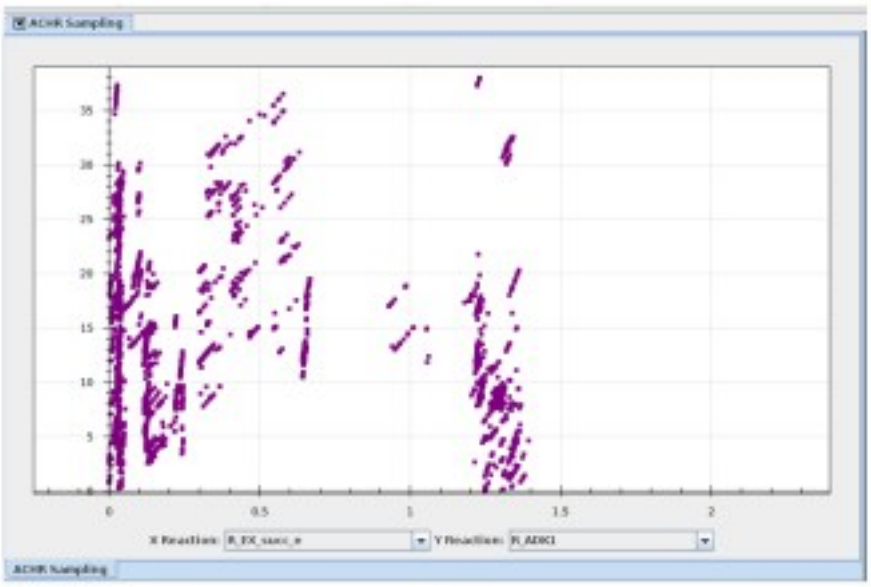

Fig. 5. The view created for the plugin. It is a XYPlot and bellow the reaction for each axis can be selected.

The view or *SamplingView*, is composed by a XYPLot. The reaction for each axis can be selected bellow, creating a plot with the flux distribution between the reactions (Figure 5). The main methods present in this methods are:

- *initPanel* - creates the view panel.
- *makeArray* - created *ArrayList* with the reaction points, retrieved from the DataType.
- *createPlot* - constructs the XYPlot.
- *updateChart* - updates the XYPlot for every new reaction selection.
- *updateCombo*, ActionListener - updates the reaction list, since the same axis cannot have the same reaction, creates an array of new points using *makeArray* and updates the chart, *updateChart*.

The *SamplingSerializer*, although not mandatory for the plugin development, it permits the user to save the sampling analysis results by creating a document and storing it in the workspace. So, every time the OptFlux software is open, its previous work is still accessible until it is deleted by the user. The *LifeCycle* class connects the constructed plugin to the OptFLux software and to change the behavior of the plugin.

3 Case Study

3.1 Introduction to *Escherichia coli* core

Several fields of Science based their studies in biological relevant species depending on their set of skills. To understand most of the cellular processes occurring in different organisms, it is often use simpler organisms, such as, *Escherichia coli*. These gram negative bacteria strains are widely use in research due to a fully sequenced genome and less intricate metabolism, which makes it easier to develop software to improve analysis and condensation of the knowledge obtained and extrapolate to other organisms. One good example is the model *Escherichia coli* core that was reconstructed based on *Escherichia coli* entire genome and containing most of the well-known reactions and fueling pathways. This model was constructed based on a single nucleotide sequence from the Escherichia coli K-12 MG1655 circular chromosome. This particular model comprises 95 reactions and 72 metabolites, where 20 are extracellular and 52 are intracellular metabolites. From those 72 metabolites only 54 are unique, as most the extracellular metabolites are derivations of its intracellular versions. The reactions are constituted by 20 exchange reactions based on the extracellular metabolites, 25 transport reactions, 49 metabolic reactions and one biomass reaction. All these reactions represent well known metabolic pathways, such as Glycolysis, Pentose Phosphate, Tricarboxylic Acid Cycle, Glycoxylate Cycle, Oxidative Phosphorylation, Fermentation and Nitrogen Metabolism [15].

3.2 Step-by-Step Tutorial

In the present case-study, we will present a step-by-step tutorial together with a small explanation of the ACHR-based Sampling Plugin developed for this model *Escherichia coli* core. A step-by-step tutorial scheme is present in Appendix B.

1. Creation of a new project. The *Escherichia coli* core model was selected from the OptFLux repository (Figure 9).
2. Initialize the ACHR Sampling. Select the Analysis >> Sampling >> ACHR Sampling from the options menu.
3. Option Selection. After the selection of the ACHR Sampling, a GUI will appear. Here the options for the initialization of the analysis can be included. The GUI already has default parameters, so the user can press enter to start the analysis. Above the GUI panel, there is an option designated as Project, where the suitable model for the data analysis is chosen and the analysis is initiated (Figure 10,11).
4. Construction of the chosen view. Once the analysis is completed a new item with the ACHR Sampling results will the added under Analysis in the OptFLux ClipBoard. After selecting it the view interface will be shown. There the user can choose the two reactions that will be used in the x axis and y axis of the view plot (Figure 12, 5).

3.3 Interpretation of the results

Once the solution space opens, the User will see the sampling plot distribution of the points between the two chosen reactions. For a better understanding of the result interpretation, several examples of plots were added in Appendix C. In the cases presented, the approximation of a line indicates correlation between both reactions, as seen in the R_GLUSy and R_GLUDy examples, whereas, the points diffusion shows no correlation between the selected reactions (Figure 16, 6).

In order to get a better perception of the points distribution, it is imperative that the number of sampling points formed during analysis increases and therefore, the point distribution will tend to the center of the solution space and progressing out of the near boundaries area. As demonstrated in the R_ICL and R_ICDHyr reactions plots, the correlation between the variables decreased when more points were considered into the analysis. Even though it continued to show some order to its distribution, the increase in the number of points considered, from 2500 to 200000 points, showed a scattering of the points obtained and therefore, hampering a definitive conclusion regarding the data correlation (Figure 14, 15). In the case of the R_EX_succ_e and R_ADK1 plots are good examples, when the distribution points are to scattered to obtain a maximum and minimum value for each reaction and thus, no correlation information can be obtained (Figure 13, 7).

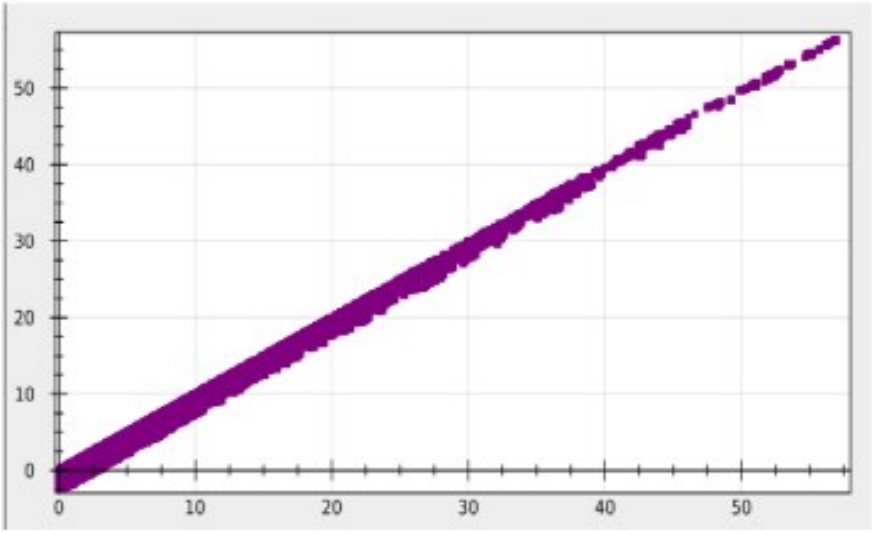

Fig. 6. R_ICL and R_ICDHyr with 200000 sampling points. X axis: right, Y axis: left.

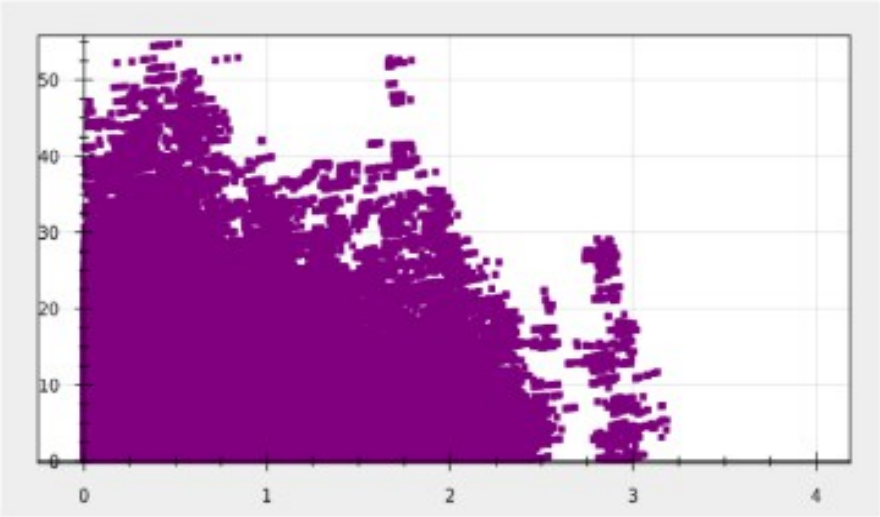

Fig. 7. R_EX_succ_e and R_ADK1 with 200000 sampling points. X axis: right, Y axis: left.

4 Conclusion

With the advent and improvement of the sequencing methods, entire genome sequencing data have been produced every year. Unfortunately, many genomes still have gaps in their sequencing data that need to be resolved, preventing in many cases to study those genes ans its enzymes at the molecular and functional level. In order to predict structure and function of these enzymes, as well as its metabolic behavior, several networks are developed in order to predict its function. To do that, constraint-based modeling is applied to these networks to simplify its complexity. Once the constraint-based model is acquired, several methods of analysis can be performed to improve results. Most analysis methods are based in the solution space boundaries while the internal fluxes re disregarded during analysis.

The sampling algorithm created in this project aimed to develop a Sampling Plugin capable of giving the best user experience when using the OptFlux, since no understanding of programming is necessary to perform or interpret the results, contrary to other commercially available software, thus, improving the data analysis. Although the plugin still requires more testing to completely execute flawlessly when results from model simulations are inserted, in case of basic sampling this plugin is capable of giving the best results. In order to improve the Plugin, I would suggest an optimization to the code could be performed so that the space and speed is reduced, and the implementation of the connection with

simulated models, such as environmental conditions, that further limitates the bounds for these reactions. Moreover, other sampling methods could be added to the OptFlux, such as the gpSampler and OptGpSampler.

References

1. Jeffrey D Orth, Ines Thiele, and Bernhard Ø Palsson. What is flux balance analysis? *Nature biotechnology*, 28(3):245–248, 2010.
2. Jan Schellenberger, Nathan E Lewis, and Bernhard Ø Palsson. Elimination of thermodynamically infeasible loops in steady-state metabolic models. *Biophysical journal*, 100(3):544–553, 2011.
3. Paulo Maia, Miguel Rocha, and Isabel Rocha. In silico constraint-based strain optimization methods: the quest for optimal cell factories. *Microbiology and Molecular Biology Reviews*, 80(1):45–67, 2016.
4. Aarash Bordbar, Jonathan M Monk, Zachary A King, and Bernhard O Palsson. Constraint-based models predict metabolic and associated cellular functions. *Nature Reviews Genetics*, 15(2):107–120, 2014.
5. Gino JE Baart and Dirk E Martens. Genome-scale metabolic models: reconstruction and analysis. In *Neisseria meningitidis*, pages 107–126. Springer, 2012.
6. Jeremy S Edwards, Rafael U Ibarra, and Bernhard O Palsson. In silico predictions of escherichia coli metabolic capabilities are consistent with experimental data. *Nature biotechnology*, 19(2):125–130, 2001.
7. Wout Megchelenbrink, Martijn Huynen, and Elena Marchiori. optgpsampler: An improved tool for uniformly sampling the solution-space of genome-scale metabolic networks. *PloS one*, 9(2):e86587, 2014.
8. Karthik Raman and Nagasuma Chandra. Flux balance analysis of biological systems: applications and challenges. *Briefings in bioinformatics*, 10(4):435–449, 2009.
9. Michael Binns, Pedro de Atauri, Anestis Vlysidis, Marta Cascante, and Constantinos Theodoropoulos. Sampling with poling-based flux balance analysis: optimal versus sub-optimal flux space analysis of actinobacillus succinogenes. *BMC bioinformatics*, 16(1):1, 2015.
10. David E Kaufman and Robert L Smith. Direction choice for accelerated convergence in hit-and-run sampling. *Operations Research*, 46(1):84–95, 1998.
11. Daniele De Martino, Matteo Mori, and Valerio Parisi. Uniform sampling of steady states in metabolic networks: heterogeneous scales and rounding. *PloS one*, 10(4):e0122670, 2015.
12. D Hyduke, D Hyduke, J Schellenberger, R Que, R Fleming, I Thiele, J Orth, et al. Cobra toolbox 2.0. *Protocol Exchange*, 2011.
13. Isabel Rocha, Paulo Maia, Pedro Evangelista, Paulo Vilaça, Simão Soares, José P Pinto, Jens Nielsen, Kiran R Patil, Eugénio C Ferreira, and Miguel Rocha. Optflux: an open-source software platform for in silico metabolic engineering. *BMC systems biology*, 4(1):45, 2010.
14. James Gary Propp and David Bruce Wilson. Exact sampling with coupled markov chains and applications to statistical mechanics. *Random structures and Algorithms*, 9(1-2):223–252, 1996.
15. Jeffrey D Orth, Ronan MT Fleming, and Bernhard Ø Palsson. Reconstruction and use of microbial metabolic networks: the core escherichia coli metabolic model as an educational guide. *EcoSal Plus*, 4(1), 2010.

Appendix A - Class Diagram of Sampling Algorithm

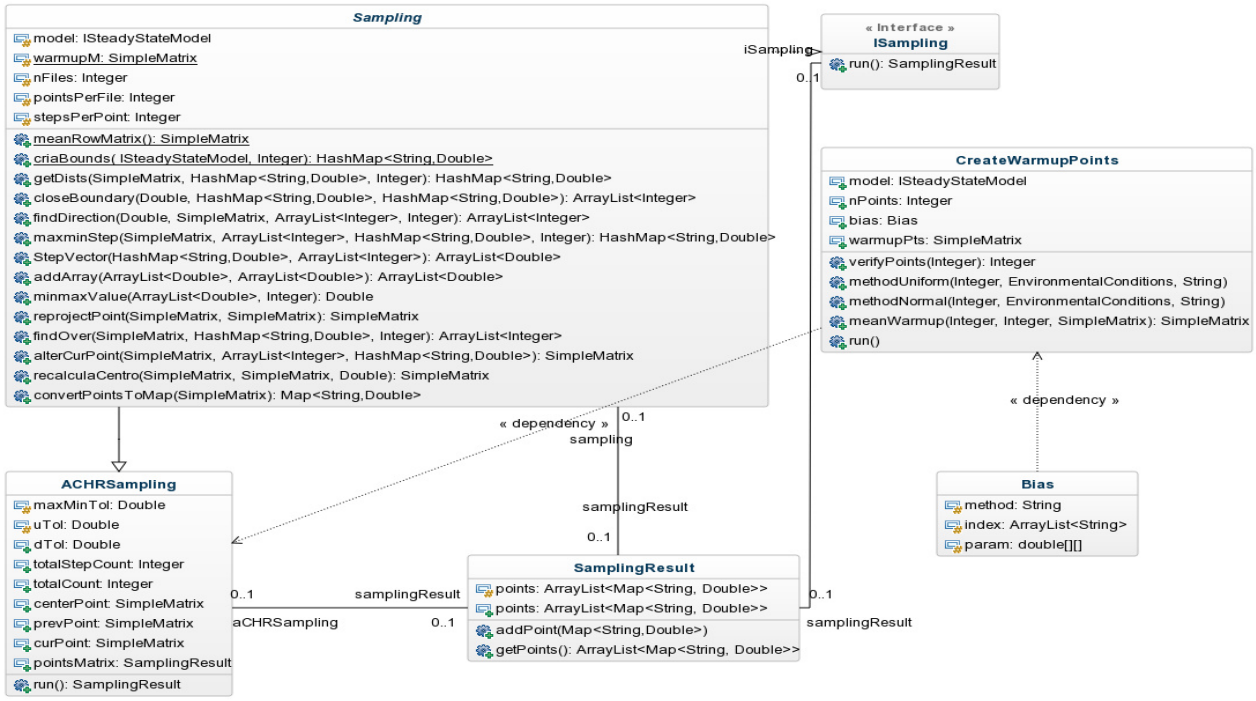

Fig. 8. Representation of the class diagram for the sampling.

Appendix B - Step-By-Step Tutorial Scheme

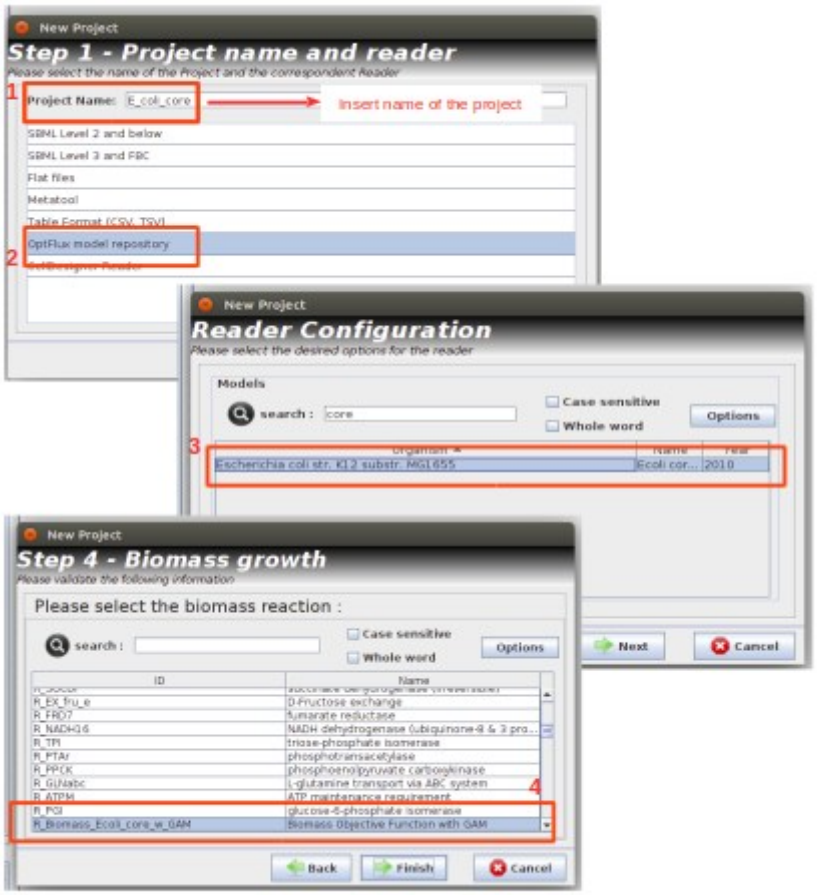

Fig. 9. Creation of a new project. The *Escherichia coli* core model was selected from the OptFLux repository.

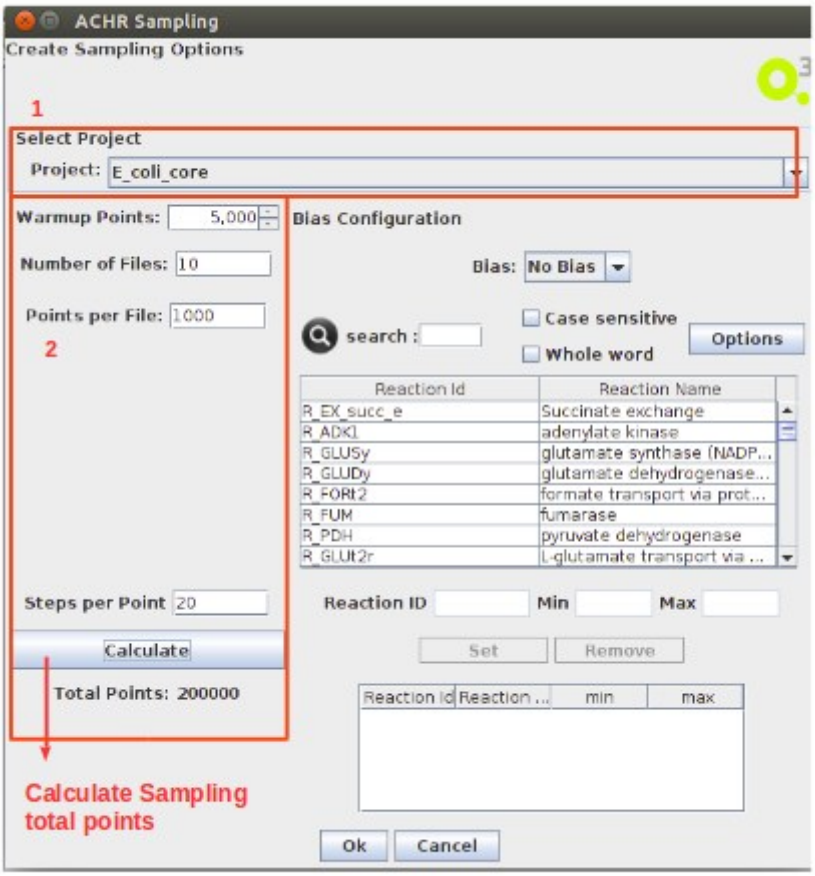

Fig. 10. Option Selection for normal sampling.

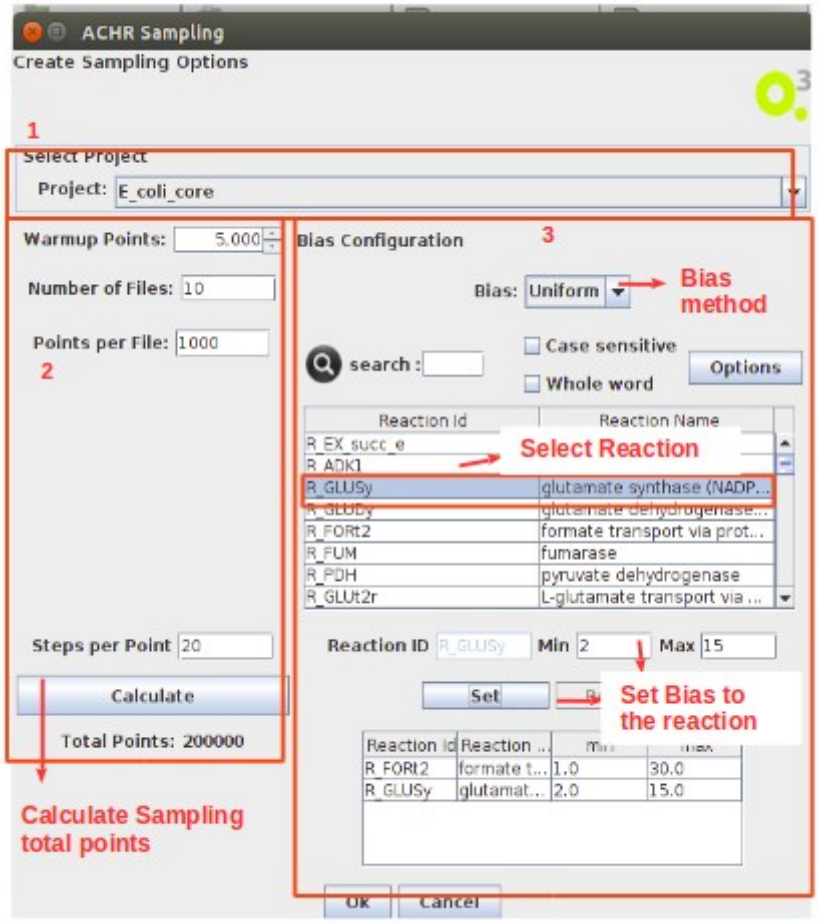

Fig. 11. Option Selection for sampling with bias option.

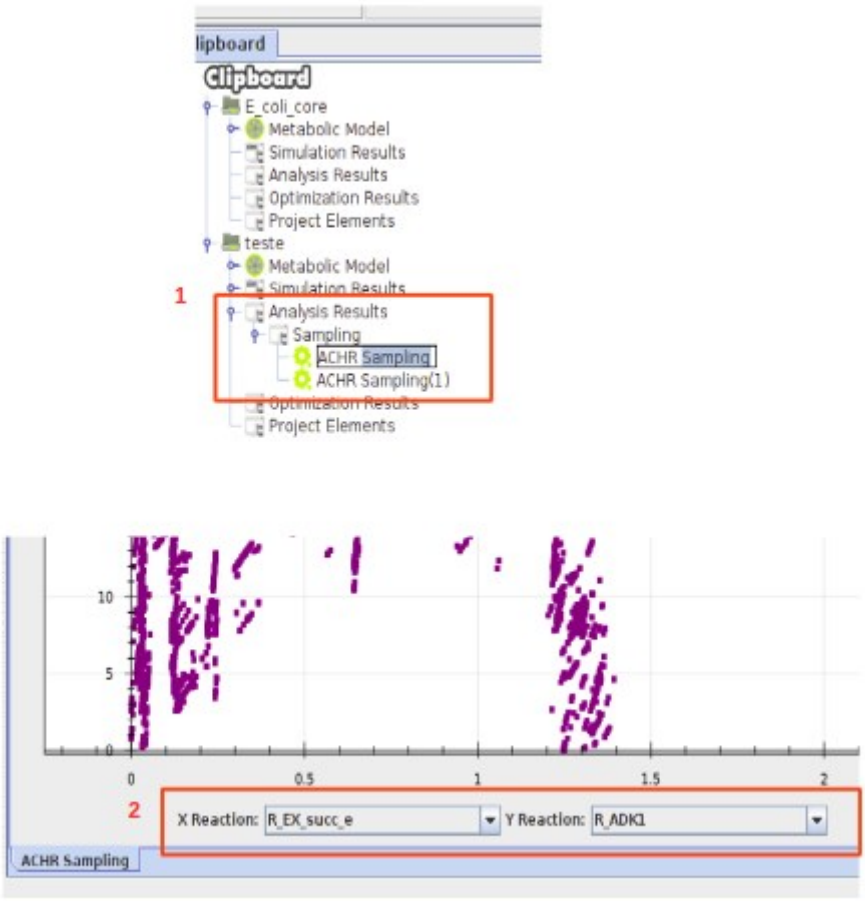

Fig. 12. Construction of the chosen view. Once the analysis is completed a new item with the ACHR Sampling results will the added under Analysis in the OptFLux ClipBoard. After selecting it the view interface will be shown. There the user can choose the two reactions that will be used in the x axis and y axis of the view plot.

Appendix C - View Results

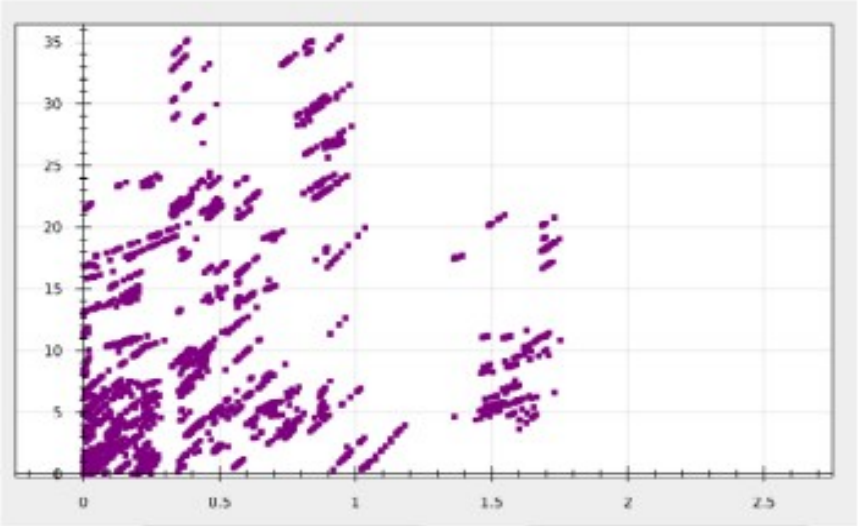

Fig. 13. R_EX_succ_e and R_ADK1 with 2500 sampling points. X axis: right, Y axis: left.

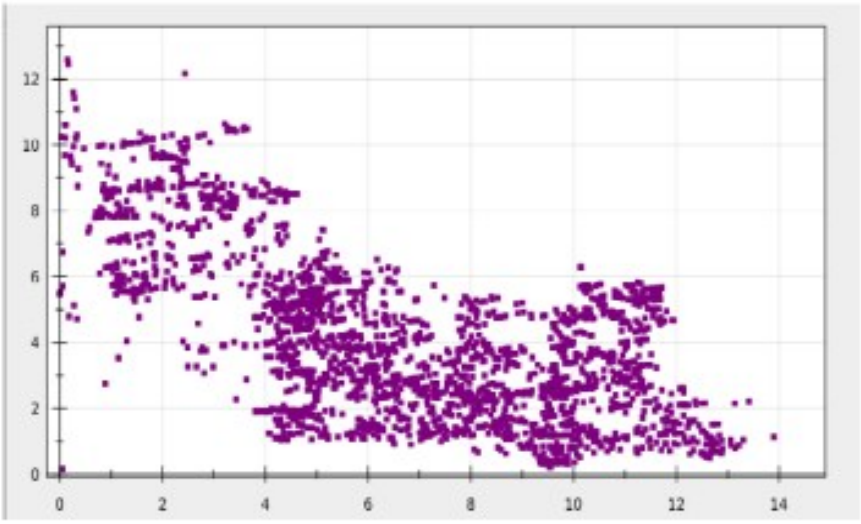

Fig. 14. R_ICL and R_ICDHyr reactions plots with 2500 sampling points. X axis: right, Y axis: left.

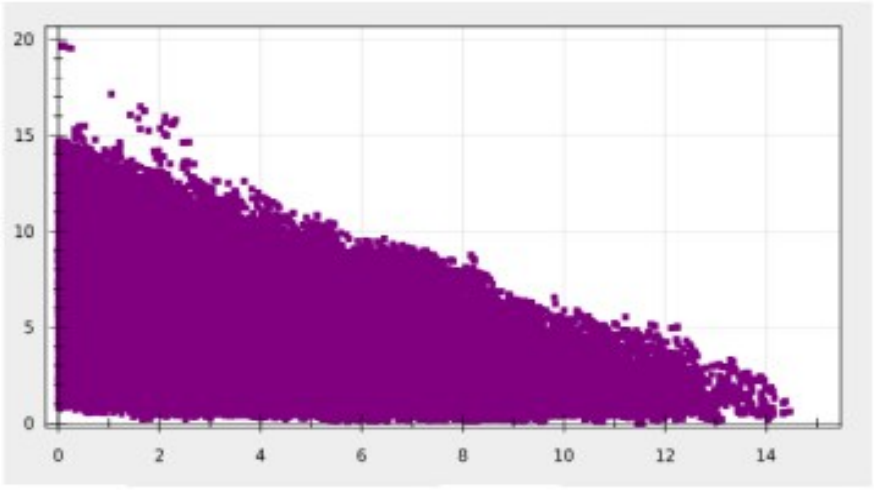

Fig. 15. R_ICL and R_ICDHyr reactions plots with 200000 sampling points. X axis: right, Y axis: left.

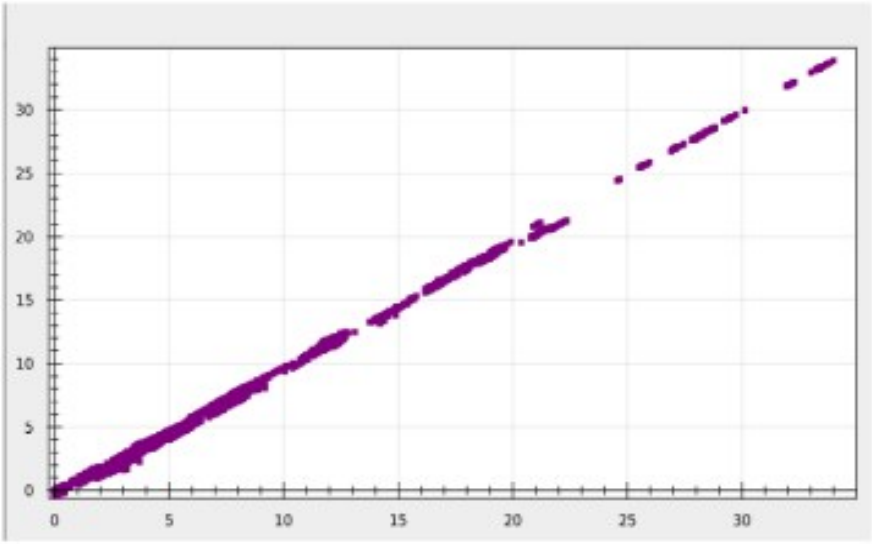

Fig. 16. R_ICL and R_ICDHyr with 2500 sampling points. X axis: right, Y axis: left.