

COBRArrow Tutorial

A tutorial for using the COBRArrow MATLAB API.

Author: Yixing Lei

INTRODUCTION

COBRArrow is a tool designed to handle large-scale biochemical reaction models through remote procedure calls (RPC). It facilitates the sharing of computational models within COBRA tools across different programming languages and provides efficient optimization services, streamlining the process of sharing and optimizing various models.

In this tutorial, you will learn how to use COBRArrow MATLAB API to:

- Set up the COBRArrow environment
- Send models to a remote server
- Fetch models from the server
- Persist models to a database
- Optimize models and retrieve results

Step 1: Environment Setup

Before you begin, ensure that you have installed the necessary dependencies:

- MATLAB with the COBRA Toolbox installed and initialized.
- Python: A compatible version of Python as specified by MathWorks Python Compatibility: <https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/support/sysreq/files/python-support.pdf>

Step 2: Setting Up COBRArrow

Before running any model, you need to set up COBRArrow in MATLAB. This involves initializing the environment and establishing a connection to the remote server.

```
% Initialize COBRArrow
initCobrarow();

% Build the service connection
client = COBRArrow('cobrarow.chatimd.org');

% Login to use the service
client = client.login('johndoe','123456');
```

Step 3: Sending and Optimizing a Model

COBRArrow allows you to send your biochemical model to a remote server, where you can run optimizations using remote resources, providing a faster and more efficient way of computation.

```
% Example: Sending a Model
% First, load your model and send it to the remote server:

% Load the model(we use ecoli_core_model for demonstration purpose)
fileName = 'coli_core_model.mat';
if ~exist('modelOri','var')
```

```

modelOri = readCbModel(fileName);
end

% Send the model to the server
schemaName = 'ecoli_core';
client.sendModel(modelOri, schemaName);

% schemaName is the unique identifier of the model on the server.
% If the model with the same schemaName already exists on server, error
% will be thrown. If you want to overwrite the existing model, set the
% toOverwrite flag to true to the method call. Or you can choose to change
% the schemaName.

% Send the model to the server with flags for persistence and overwrite
toPersist = false;           % Persist the model in the database
toOverwrite = true;          % Overwrite the model on the server if it exists
client.sendModel(modelOri, schemaName, toPersist, toOverwrite);

% or you can just call sendModel method like this:
client.sendModel(modelOri, schemaName, false, true);

% Example: Optimizing a Model
% Once the model is on the server, you can run optimization on the server.

% Specify the solver and parameters
solver = COBRArrowSolver('gurobi');
solver.setParameter('tol', 1e-9);
% Note: currently the remote optimization service hasn't supported
% parameter change yet

% Run optimization
result = client.optimizeModel(schemaName, solver);

```

Additional Methods: Fetching and Persisting Models

1. Fetching the Model

You can fetch a model that was uploaded by other users. The model can be used in COBRA Toolbox for further analysis.

```

fetchedModel = client.fetchModel(schemaName);

% For example, it can be used to do optimization locally.
solution = optimizeCbModel(fetchedModel);

% 2. Persisting the Model to a Database
%
% Models sent to the remote server are stored in memory, which facilitates
% fast data sharing. However, this data is volatile and may be lost if the
% server is abruptly terminated.

```

```

% To avoid losing data, it's important to persist the model to a database:
client.persistModel(schemaName, true);

% 3. Send a field
%
% You can send an individual field to server.
% For example, the flux result can be sent to server for visualization in
% Escher map.
fieldName = 'flux';
fieldData = result.flux;
client.sendField(schemaName, fieldName, fieldData);

% If the field is already stored on the server, error will be thrown. You
% need to set the toOverwrite flag as true to overwrite the data.
client.sendField(schemaName, fieldName, fieldData, true);

% 4. Fetch a single field.
%
% You can fetch a single field in the model from the server. This provides
% efficiency and convenience when you only need part of the data in the
% model for analysis.
fluxResult = client.fetchField(schemaName, fieldName);

% 5. Fetch the model for FBA analysis
%
% You can fetch part of the model which includes the fields that are used
% for doing FBA analysis only.
FBAmode1 = client.fetchModelForFBAAnalysis(schemaName);
FBAr1esult = optimizeCbModel(FBAmode1);

% 6. List flight information on the server
%
% The model sent to the server is stored as many flights. Every flight
% contains information about how each field is stored on the server. It
% includes:
% - descriptor: Unique identifier for the flight.
% - total_records: Total number of records in the flight.
% - total_bytes: Total number of bytes in the flight.
% - endpoints: List of endpoints where the flight can be accessed.
% - schema: Schema of the flight, including column names and data types.
% - schema_metadata: Additional metadata associated with the schema, such
% as field name, description, dimension of the field and the data type of
% the field in matlab.

% List flights in a schema
flightsList = client.listAllFlights(schemaName);

% List all flights on server
allFlightsList = client.listAllFlights();

```

```

% 7. Get unique identifier of the flights
%
% Get unique identifiers of the flights in a schema, so that we can
% retrieve the flight and get the data.
descriptors = client.getAllDescriptors(schemaName);

% Get all unique identifiers of the flights on the server
allDescriptors = client.getAllDescriptors();

```

TROUBLESHOOTING

If you encounter issues with sending models or running optimizations, ensure: 1. The COBRA Toolbox and dependencies are correctly installed and configured. 2. The server is accessible and running the necessary services. 3. Python and required package (PyArrow) is correctly installed.

Acknowledgments

1. Currently, the remote optimization service supports solvers including: CPLEX, GLPK, Gurobi. It hasn't supported parameter change yet. 2. Always persist your model to the database to avoid losing your data in case of a service shutdown. 3. The listAllFlights method and getAllDescriptors method can be used for debugging and future extension purposes, they are not involved in general use cases.