MARS Network Services User Manual
Version 1.0

Sherif Abdelhamid, Chris Kuhlman

December 9, 2016

Contents

Abstract

The purpose of this document is to provide an overview of MARS services, installation and usage.

1 Introduction

1.1 What is MARS?

MARS is a new network services and workflow system that supports structural network analyses and generalized network dynamics analyses. It is accessible through the internet and can serve multiple simultaneous users and software applications. In addition to managing various types of digital objects including networked data, MARS provides services that enable applications (and UIs) to add, interrogate, query, analyze, and process data.

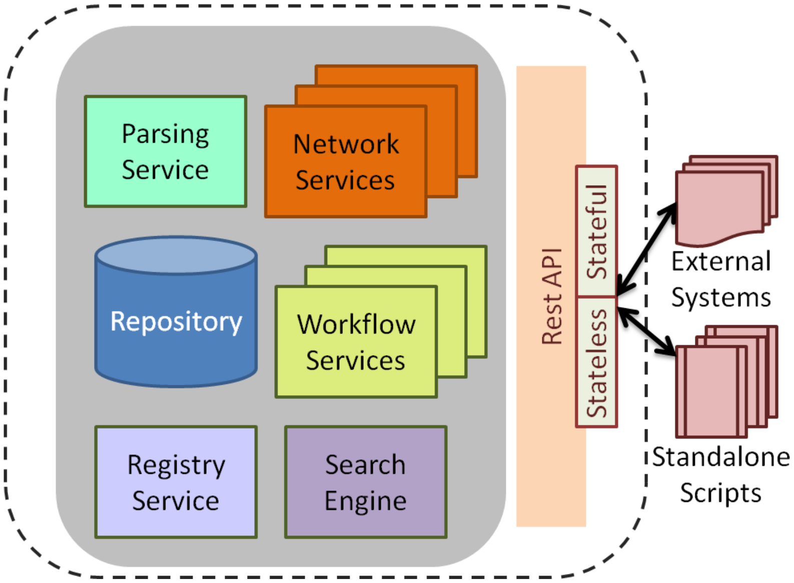


Figure 1: Overview of the MARS system.

1.2 Features

- Modular, interoperable categories of services, where each service can be a distinct, relocatable process.
- Stateless (REST-ful) and stateful (using session management) API.
- SQL-like query grammar with special features for network dynamics.
- Built-in search engine, for quey grouping, sharing and retrieval.
- Database for storing networked data along with metadata.

---

2 MARS Setup

2.1 Installation

Currently, MARS v1.0 is tested to run on Linux platform. To install MARS:

2.1.1 Python Installation

- wget https://www.python.org/ftp/python/2.7.9/Python-2.7.9.tgz
- tar -xzvf Python-2.7.9.tgz
- cd Python-2.7.9
- ./configure –prefix=[Python installation directory]
- make
- make test
- make install

2.1.2 Installing Required Packages

- Check if pip or easy_install are installed, if not do:
  - wget https://bootstrap.pypa.io/get-pip.py
  - python get-pip.py
  - wget https://bootstrap.pypa.io/ez_setup.py −o − | python
  - easy_install pip
- Use pip or easy_install to install the packages in Table 1 (e.g. easy_install numpy or pip install whoosh):

Table 1: List of python packages used for MARS implementation.

Package	**Purpose**	**Source**
SQLITE	Data Repository	https://sqlite.org/
Cherrypy	Multi-threaded python server	http://cherrypy.org/
Whoosh	Search Engine	https://pypi.python.org/pypi/Whoosh/
Pyparsing	Parser Generator	http://pyparsing.wikispaces.com/
Bottle Framework	Rest API	http://bottlepy.org/docs/dev/
MPipe	User-defined Workflows	http://vmlaker.github.io/mpipe/
matplotlib	Data Plotting	http://matplotlib.org/
pythonds	Internal Data Structures (Stack)	https://pypi.python.org/pypi/pythonds
requests	Send/Receive HTTP Requests	http://docs.python-requests.org/en/master/
NumPy	Data Analysis	http://www.numpy.org/
NetworkX	Network Structure Measures	https://networkx.github.io/

2.1.3 Downloading MARS Code

- Ask Sherif Abdelhamid (sherief@vbi.vt.edu) or Chris Kuhlman(ckuhlman@vbi.vt.edu) to be added to git project (https://ndsslgit.vbi.vt.edu/software-contagion-services/network-services)
- enter git clone https://ndsslgit.vbi.vt.edu/software-contagion-services/network-services.git
- cd network-services
- check the main directories v1 and v2. Each directory has two sub-directories src (code) and doc (manual).
- v1.0 codes resides under v1/code sub-directory.
- Copy database file into database directory of your choice (This is a database directory that you created or have access permission to it. Please make sure that it doesn't contain a database file with the same name).
  scp username@edisondev.vbi.vt.edu:/home/sipcinet/edison/graphservices/database/Edison2.db [database directory]

2.1.4 configuring MARS services

- go to [code directory]/v1/src/code
- check and edit (if needed) the properties:
  - server: name of the python server used. Currently, set to cherrypy, a multithreaded server written python.
  - host: server IP address that will host MARS. It can be set to localhost.

- port: list of port numbers that will be used. Each port number is in a separate line. Two services can not share the same port. Make sure when add port to the config file that only one service can use at a time.
- database: the database directory.
- index1: location where property queries are indexed on the file system.
- index2: location where seed queries are indexed on the file system.
- query: directory where query results are stored on the file system.
- graph: directory for MARS graphs.
- code: the directory that contains the services code and the stand-alone executable codes that compute measures on networks. These codes can be C, C++, Python, Perl, or codes written in any other language. These are called by the measure service in MARS v2.

2.1.5 Starting MARS Services on edisondev

Note: the same steps can be done on other VMs, as long as the user has the required permissions and access rights.

- sudo su - sipcinet
- export PATH=[python installation directory]:$PATH
- go to [code directory]/v1/src/code
- enter python query_service.py &

2.1.6 Starting MARS Services on edison VM (edison.vbi.vt.edu)

(Note: the same steps can be performed on other VMs as long as the correct directories are provided)

- sudo su - sip
- export PATH=[python installation directory]:$PATH
- go to [code directory]/v1/src/code
- enter python query_service.py &

2.1.7 Stopping MARS Services

- ps ax | grep python [Retrieve python process PID]
- kill −9 PID

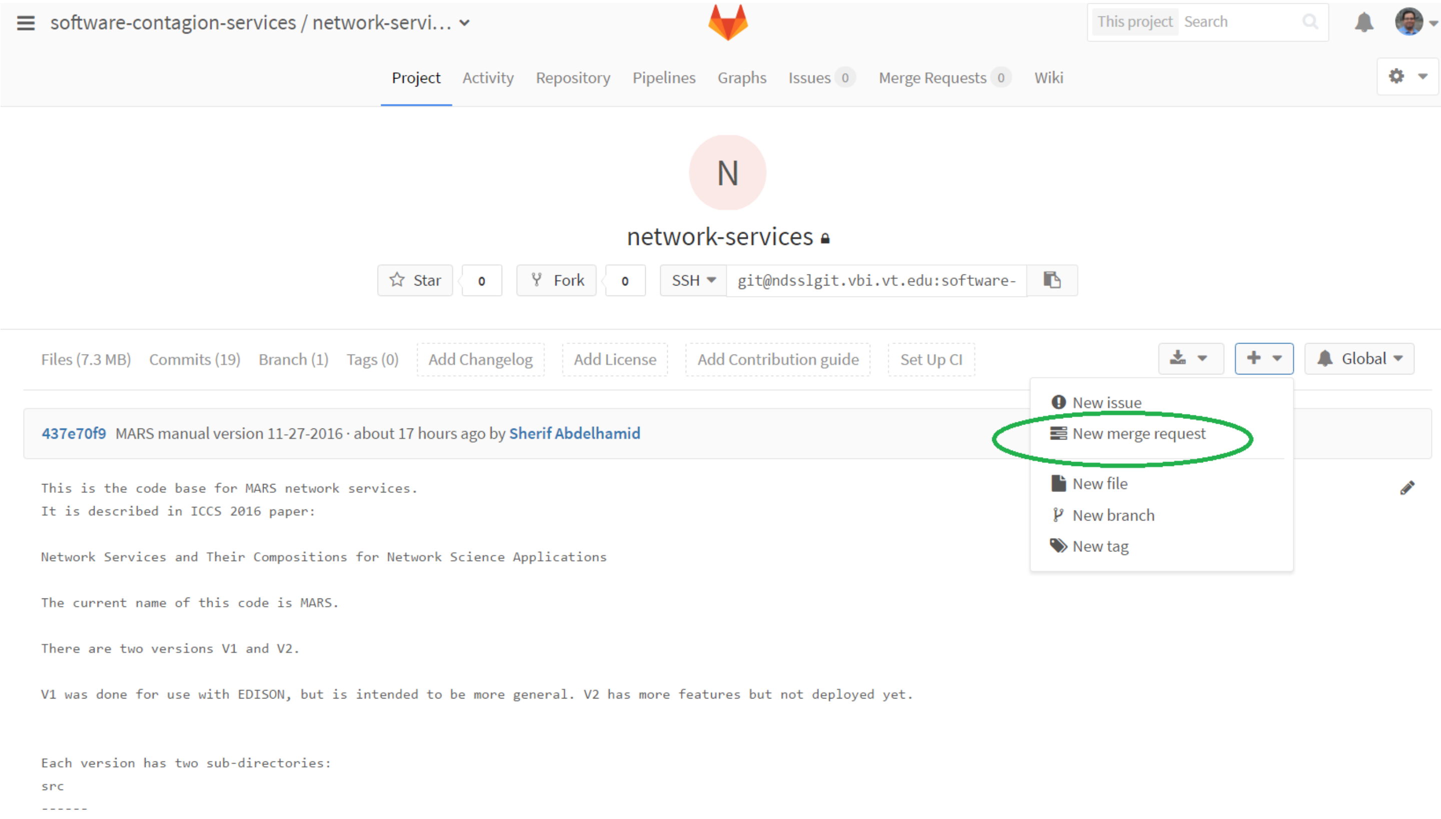
2.1.8 Create index for Network Query Search Service

- go to [code directory]/v1/src/code
- enter python create_index.py [This will create two indexes, for the property and seed type queries]. The indexes directories are specified in the mars.config file as index1 and index2.

2.2 Contribution steps

To start contributing to MARS repository, please follow these steps:

- Create a new branch and switch to it [git checkout –b [name_of_your_new_branch]. Name of branch should begin with class type:
  - Feature, if user is adding a new feature for MARS.
  - Bug-fix, if user updating current code to fix a bug.
  - Enhancement, if user is updating current code for code optimization or performance enhancement.
- Commit the changes with a descriptive message. [git commit -m "Commit message"].
- Create a new remote for your branch [git remote add [name_of_your_remote]].
- Push the branch to the remote repository [git push [name_of_your_new_remote] [name_of_your_new_branch]].
- Create a merge request using https://ndsslgit.vbi.vt.edu, as shown in Figure 2. Notify other team members to review.
- Reviewer will check for merge conflicts.
- Reviewer merge the new branch with master.



437e70f9 MARS manual version 11-27-2016 · about 17 hours ago by Sherif Abdelhamid

```
This is the code base for MARS network services.
It is described in ICCS 2016 paper:

Network Services and Their Compositions for Network Science Applications

The current name of this code is MARS.

There are two versions V1 and V2.

V1 was done for use with EDISON, but is intended to be more general. V2 has more features but not deployed yet.


Each version has two sub-directories:
src
------
```

Figure 2: Creating a merge request using Web-based interface of ndsslgit.

3 Using MARS

Currently, MARS v1.0 is used with EDISON system. However, MARS is a general purpose network services that can be integrated with other systems through the REST API. In this section we describe all MARS endpoints grouped by each service. A sample network, see Figure 3, is used in the illustrative examples in the API document. Each node has eight attributes (id integer, degree integer, betweeness_centrality real, clustering real, load_centrality real, node_clique_number integer, closeness_centrality real, clustering_galib real).
Notes:

- All the possible responses are listed under 'Responses' for each method. Only one of them is issued per request server.
- All responses are in JSON format.
- Some of requests parameters are optional and have default value.
- Host name and port number can be changed in mars.config file.

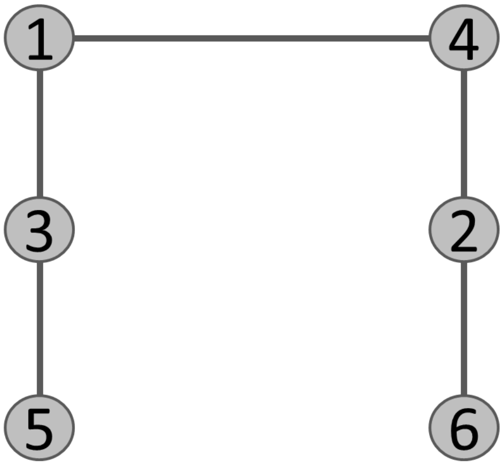


Figure 3: Sample network used in the illustrative examples. The network consists of six nodes. All the nodes have degree equals to two, except nodes 5 and 6 which have degree equals to one.

3.1 MARS API Document

Network Query Service

Methods

1. Query

Purpose

Submit queries to the Network Query Service.

Request

Method	URL
GET	`http://<hostName>:<portNumber>/graphservice/query?content=value&graph=value&validate=value&view=value&parallel=value&nruns=value`

Type	Params	Values	Examples	Required	Description
GET	content	string	1. select nodes from netscience where degree > 20 2. select edges from netscience where coauthorship> 3 3. select sample(5,[3,8]) nodes from dolphins	Yes	Query sting: two forms simple (examples 1 and 2) and sampling (example 3).
	graph	string	netscience	Yes	graph name
	validate	string	True/False	No (default False)	Valid entries are True or False only
	view	string	property/seed	Yes	Valid entries are either property (for simple queries) or seed (for sampling queries).
	parallel	string	True/False (Please always use False, feature not supported in the deployed version)	No (default False)	Valid entries are True or False only
	nruns	string	3	No (default0)	Used only if seed view is selected

Response

Status	Response

200	**Response for simple query will be an object containing the list of nodes or edges as well as the attributes.**
	Note **that the WHERE clause in queries is disabled in this version of MARS in order to proces queries from the current version of the EDISON application. Currently, the queries invoked from the EDISON application must return all nodes or edges, not subsets of them.**
	
	An example response returning all nodes (100% coverage):-
	`select nodes from sample`
	
	{"node_sets": [{"nodes": [{"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.555555555556, "clustering_galib": 0.0, "load_centrality": 0.6, "id": 1, "betweeness_centrality": 0.6}, {"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.454545454545, "clustering_galib": 0.0, "load_centrality": 0.4, "id": 2, "betweeness_centrality": 0.4}, {"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.454545454545, "clustering_galib": 0.0, "load_centrality": 0.4, "id": 3, "betweeness_centrality": 0.4}, {"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.555555555556, "clustering_galib": 0.0, "load_centrality": 0.6, "id": 4, "betweeness_centrality": 0.6}, {"clustering": 0.0, "node_clique_number": 2, "degree": 1, "closeness_centrality": 0.333333333333, "clustering_galib": 0.0, "load_centrality": 0.0, "id": 5, "betweeness_centrality": 0.0}, {"clustering": 0.0, "node_clique_number": 2, "degree": 1, "closeness_centrality": 0.333333333333, "clustering_galib": 0.0, "load_centrality": 0.0, "id": 6, "betweeness_centrality": 0.0}], "coverage": 100.0}]}
	
	An example response returning all edges in karate network (100% coverage)
	
	`select edges from sample`
	
	{"edge_sets": [{"edges": [{"start": 2, "end": 4}, {"start": 3, "end": 1}, {"start": 3, "end": 5}, {"start": 2, "end": 6}, {"start": 1, "end": 4}], "coverage": 100.0}]}

	Response for sampling query. **This query returns three sets of edges, this is identified by the number “2” after the “number =” keyword. Each set may contain any number of nodes from 1 up to 2, this is shown in “[1,2]” after the keyword “size =”.** **Note here that we used a where clause that is acceptable in seed type queries only.** `select sample(number = 2,size =[1,2])nodes from sample where degree > 1`

	{"node_sets": [{"nodes": [{"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.454545454545, "clustering_galib": 0.0, "load_centrality": 0.4, "id": 3, "betweeness_centrality": 0.4}, {"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.555555555556, "clustering_galib": 0.0, "load_centrality": 0.6, "id": 1, "betweeness_centrality": 0.6}], "coverage": 33.3}, {"nodes": [{"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.454545454545, "clustering_galib": 0.0, "load_centrality": 0.4, "id": 3, "betweeness_centrality": 0.4}, {"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.555555555556, "clustering_galib": 0.0, "load_centrality": 0.6, "id": 1, "betweeness_centrality": 0.6}], "coverage": 33.3}]}
	This query returns three sets of nodes, this is identified by the number "3" after the "number =" keyword. Each set contains exactly one node, this is shown in "[1,1]" after the keyword "size =". Note here that we didn't use the where clause which is optional.
	`select sample(number = 3,size =[1,1])nodes from sample`
	{"node_sets": [{"nodes": [{"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.454545454545, "clustering_galib": 0.0, "load_centrality": 0.4, "id": 3, "betweeness_centrality": 0.4}], "coverage": 16.7}, {"nodes": [{"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.454545454545, "clustering_galib": 0.0, "load_centrality": 0.4, "id": 3, "betweeness_centrality": 0.4}], "coverage": 16.7}, {"nodes": [{"clustering": 0.0, "node_clique_number": 2, "degree": 2, "closeness_centrality": 0.555555555556, "clustering_galib": 0.0, "load_centrality": 0.6, "id": 4, "betweeness_centrality": 0.6}], "coverage": 16.7}]}

200	**Response for query validation** {"check": "Valid"}
400	"Invalid Query"
300	"Do you mean degree" if user misspelled attribute name, service responds with the closest attribute suggestion

3. Search

Purpose

Search for existing queries. For example, in the EDISON web application, this invocation is used to return all queries that have been executed, so that they may be displayed in the UI, to help users form their own queries or use pre-existing ones.

Request

Method	URL
GET	`http://<hostName>:<portNumber>/graphservice/search?keywords=value&metadata=value&view=value`

Type	Params	Values	Examples (Valid entries)	Required	Description
GET	keywords	String	sample NOT degree	yes	Keywords used for query searching. Users can use Boolean operators as an example AND, OR, NOT
	metadata	string	content/graph/query_id/target	No (Default content)	Type of metadata searching is based on. Valid entries are only content, graph, query_id or target. “query_id” is an integer number that uniquely identifies the query. “target” is the target type which can be node or edge. “graph” is the name of the network on which the query is executed.
	view	string	property/seed	yes	The type of queries. Valid entries are property or seed only.

Response

Status	Response
200	**Response will be an object containing the list of queries (array).** **Using the keywords sample NOT degree which means search for all the queries that have the keyword sample but not degree.** [{"content": "select edges from sample", "graph": "sample", "query_id": "2530", "results": "/home/sipcinet/edison/graphservices/query/sample_query2530.txt", "target": "edge"}, {"content": "select nodes from sample", "graph": "sample", "query_id": "2527", "results": "/home/sipcinet/edison/graphservices/query/sample_query2527.txt", "target": "node"}]
400	`"Invalid Search"`

4. Repository

Purpose

Repository keeps information about all queries. Categorize queries by node and edge.

Request

Method	URL
`GET`	`http://<hostName>:<portNumber>/graphservice/repository?type=value&view=value`

Type	Params	Values	Examples	Required	Description
`GET`	type	string	node/edge/ all	yes	Category of data to be retrieved. Valid entries are node, edge or all only
	view	string	property/seed	yes	Type of queries. Valid entries are property or seed only.

Response

Status	Response
200	**An example response for choosing type node and view seed queries. This will return all queries targeting nodes of type seed for sampling.** {"node_queries": [{"content": "select sample(number = 3,size =[1,1])nodes from sample", "graph": "sample", "query_id": "2532", "results": "/home/sipcinet/edison/graphservices/query/sample_query2532.txt", "target": "node"}, {"content": "select sample(number = 2,size =[1,2])nodes from sample where degree > 1", "graph": "sample", "query_id": "2531", "results": "/home/sipcinet/edison/graphservices/query/sample_query2531.txt", "target": "node"}]} **An example response for choosing type node and view property queries. This will return all queries targeting nodes of type simple.** {"node_queries": [{"content": "select nodes from sample where degree = 1", "graph": "sample", "query_id": "2529", "results": "/home/sipcinet/edison/graphservices/query/sample_query2529.txt", "target": "node"}, {"content": "select nodes from sample where degree >2" "graph": "sample", "query_id": "2528", "results": "/home/sipcinet/edison/graphservices/query/sample_query2528.txt", "target": "node"}, {"content": "select nodes from sample", "graph": "sample", "query_id": "2527", "results": "/home/sipcinet/edison/graphservices/query/sample_query2527.txt", "target": "node"}]}
400	`“Invalid Input"`

Network Storage Service

Methods

1. Graph

Purpose

List all stored graphs. Search a graph by name or filter graphs by attribute value

Request

Method	URL	Description
GET	http://<hostName>:<portNumber>/graphservice/torage/graph	This end point returns all graphs marked as available
GET	http://<hostName>:<portNumber>/graphservice/torage/graph/filter?attribute=value&operatorvalue&rval=value	This end point returns all graphs marked as available and satisfy the given filter. Filter is defined by attribute, operator and value

Type	Params	Values	Example	Description
GET	attribute	string	network attribute	Network attributed used for filtering. Network attributes are part of network metadata.
	operator	string	>,>=, <, <=, !=,=	Operator used for the comparison. Valid values are >,>=, <, <=, !=,=
	rvalue	string	500	Right-hand value. This can be sting or number. Has to be consistent with the attribute data type. For example, if the attribute on left-hand side is of type sting, the rvalue shall be sting too.

Response

Status	Response
200	**Response will be a JSON object containing a single graph or a list of graphs with details, including title, description and other metadata. Here we sent a request using attribute nodes, operator <, and rvalue 40. This will return all graphs in the repository that are marked available to public and have number of nodes < 40.** [{"directed": "false", "weighted": "false", "graph_id": 38, "name": "karate", "edge_attributes": {"degree_product": "integer", "betweenness_centrality": "real"}, "numberOfEdges": 78, "file_name": "karate", "original_format": "uel", "labeled": "true", "node_attributes": {"node_clique_number": "integer", "closeness_centrality": "real", "degree": "integer", "betweenness_centrality": "real", "load_centrality": "real", "id": "integer", "clustering": "real"}, "numberOfNodes": 34, "description": "Network of friendships between the 34 members of a karate club at a US university, as described by Wayne Zachary in 1977"}]
400	“Database Error”

Model Information Service

Note: This service is in not use now, and will take over in the future all model-related processes from Edison front end.

Methods

Purpose

List all Edison models with information

Request

Method	URL
GET	http://<hostName>:<portNumber>/graphservice/model

Response

Status	Response
200	**Response will be a JSON object containing the a list of models with details, including description, parameters and other metadata** [{"sub_model_id": 1, "model_id": 11, "model_name": "Progressive 2-state model", "parameters": [{"threshold": "integer", "model": "integer", "type": "int_node_trait", "sub_model": "integer"}, {"state": "integer", "is_fixed": "integer", "type": "int_node_state"}], "description": "Threshold model where nodes may transition from state 0 t 1, but not from 1 to 0. The model supports blocking nodes: nodes that do not change state."}, {"sub_model_id": 3, "model_id": 11, "model_name": "Back-and-forth 2-state model", "parameters": [{"type": "int_node_trait", "model": "integer", "down_threshold": "integer", "sub_model": "integer", "up_threshold": "integer"}, {"state": "integer" "is_fixed": "integer", "type": "int_node_state"}], "description": "Threshold model where nodes may transition from state 0 to 1, and from 1 to 0. The model supports blocking nodes: nodes that do not change state."}, {"sub_model_id": 4, "model_id": 11, "model_name": "Back-

	and-forth 2-state model with influence from distance-2 neighbors", "parameters": [{"type": "int_node_trait", "model": "integer", "down_threshold": "integer", "sub_model": "integer", "up_threshold": "integer"}, {"state": "integer", "is_fixed": "integer", "type": "int_node_state"}], "description": "Threshold model where nodes may transition from state 0 to 1, and from 1 to 0. Neighboring nodes at distance 1 and distance 2 can influence a node."}, {"sub_model_id": 1, "parameters": [{"model": "integer", "duration_in_state_I": "integer", "type": "int_node_trait", "sub_model": "integer"}, {"state": "integer", "type": "int_node_state"}, {"edge_weight": "double", "type": "double_edge_state"}], "model_name": "SIR epidemic model", "model id": 22, "description": "Classic susceptible-infected-recovered epidemiological model."}, {"sub_model_id": 3, "model_id": 22, "model_name": "SIR epidemic model", "parameters": [{"model": "integer", "type": "int_node_trait", "sub_model": "integer", "duration_in_state I": "integer"}, {"threshold": "integer", "state": "integer", "type": "int_node_state"}, {"edge weight": "double", "type": "double_edge_state"}], "description": "Classic susceptible infected-recovered epidemiological model, but a susceptibl node may require multiple infecting neighbors to become infected."}, {"sub_model_id": 0, "model_id": 37, "model_name": "Linear Threshold model", "parameters": [{"model": "integer", "type": "int_node_trait", "sub_model": "integer"}, {"state": "integer", "type": "int_node_state"}, {"threshold": "double", "type": "double_node_state"}, {"type": "double_edge_state", "edge influence": "double"}], "description": "This is the Linear Threshold model of Kempe et. al (KDD 2003)."}, {"sub_model_id": 0, "parameters": [{"model": "integer", "type": "int_node_trait", "sub_model": "integer"}, {"state": "integer", "is_fixed": "integer", "type": "int_node_state", "down_threshold": "integer", "up_threshold": "integer"}], "model_name": "Connected Components Threshold Model", "model id": 38, "description" "This is an influence model that uses thresholds, but now each set of neighbors of a node that form a connected component collectively influence the node (Ugander, PNAS 2012)."}]

4 Adding New Networks Manually

This section illustrates by example, the steps needed to add a new network to MARS repository. The same steps can be applied to larger networks. Users can create their own shell, python, or R scipts to automate these steps.

Note: We will use network "sample" for demonstration.

Files needed:

File	Description
node_file_gen.py	Generates uel, del and nodes file from input file
gen_n_file.qsub	Executes node_file_gen.py on sfx
preparefiles.py	Generates uel and node files with attributes for database
preparefiles.qsub	Executes preparefiles.py on sfx
dataLoader.py	Loads network data from file into database
Loaddata.qsub	Executes dataLoader.py on sfx through qsub

Processing and generating data on shadowfax:

1. Go to you scripts directory under v1/doc/manu01
2. Edit gen_n_file.qsub and replace graph name with sample2.
3. Execute qsub gen_n_file.qsub on sfx1, this will generate: sample.uel, sample.del and sample.nodes. These files are needed by InterSim.
4. Edit preparefiles.qsub and replace graph name with sample2. Note: make sure the path for the preparefiles.py is correct.
5. Edit preparefiles.py and replace all directories for .uel, .nodes, .uels, and .info files with the correct ones.
6. Execute preparefiles.qsub on sfx1, this will generate: sample2.info and sample2.uel2.

Creating database entries

1. Go to the database directory.
2. Type sqlite3 Edison2.db, this will connect you to Edison database.
3. You can rename database file to any other name if needed.
4. Type .schema to get info about current tables and indexes in the database.
5. type .schema <table name> to get information about specific table
6. Now we will create two tables for the new network: sample_node and sample_edge.
7. The number and data type of the new tables' attributes should be consistent with data columns in the two newly generated files sample2.info and sample2.uel2.
8. After creating the two tables. type .schema sample_node to make sure table is created successfully. Do the same step again .schema sample_edge.
9. Now to add the metadata for the network, we use table network. To get familiar with existing networks type select * from network. Available networks to Edison have the attribute “available” set to true.

10. Before changing/ adding data in network table, dump existing data to a save place as backup. To do these steps:

```
sqlite> .mode csv
sqlite> .output network_backup_4_7_2016.csv
sqlite> select * from network;
sqlite> .output stdout
```

11. Using sql insert command enter the new network data, don't include id as it is automatically generated in the dbms.
12. all networks should have available attribute set to false at beginning until they pass all testing.

Importing data into database

1. Go to scripts directory. Edit loaddata.qsub file, put the name of the table first, in this case sample. Followed by name of node and edge attributes files sample2.uel and sample2.info. This will load the data from each file to the corresponding table. Data includes node/edge ids and attributes.
2. To verify data is loaded correctly:
    a. login back to database
    b. verify number of nodes correct "select count(*) from sample_node" and compare with network number of nodes.
    c. verify number of edges correct "select count(*) from sample_edge" and compare with network number of edges.
    d. verify if isolated nodes exist "select count(*) from sample_node where id not in (select start from sample_edge) and id not in (select end from sample_edge)" if count == 0 then no isolated nodes exist.
    e. verify if network is undirected or has no duplicate edges "select count(*)/2 as duplicates from sample_edge a, sample_edge b where a.start = b.end and a.end = b.start" if duplicates == 0 then no duplicate edges exist.
    f. verify if self loops exists "select count(*) from sample_edge a where start == end " if count == 0 that means no self loops.
    g. verify if node ids are not negative "select count(*) from temp_node where id < 0 " and "select count(*) from temp_edge where start < 0 or end < 0" count should be 0 in both cases.
    h. verify if nodes ids are numeric select count(*) from sample_edge where typeof(start) = "integer" and typeof(end) = "integer" and select(*) from sample_node where typeof(id) = "integer"