Tutorial on Recast: generating graphs from existing traces

1) Create a folder named “traces” on your “workspace” directory. All your traces should be there. To check if your code is pointing to this folder, on DB.java, check the line:

```
public final static String DB_FOLDER = "..\\traces\\";
```

2) Create a folder named “Graphs_Data” on your “workspace” directory. In this folder, create a folder for each trace you are using. On DB.java, check the code:

```
public static String[] DB_GRAPHS_PATH = {
	"../Graphs_Data/Dart/",
	"../Graphs_Data/Cambridge/",
	"../Graphs_Data/Taxi/",
	"../Graphs_Data/Infocom/",
	"../Graphs_Data/USC/",
	"../Graphs_Data/NUS/",
	"../Graphs_Data/UPB/",
	"../Graphs_Data/SASSY/",
	"../Graphs_Data/4SQUARE-NY/",
	"../Graphs_Data/4SQUARE-TK/",
	"../Graphs_Data/4SQUARE-SP/"};
```

3) With this code, you can

1. Generate encounter graphs from the traces
2. Simulate data dissemination from the traces and from the RECAST classes

All the possible ways to run the code are listed in class RunScripts.java.

4) In order to generate graphs, you have to call the procedure `runGenerateGraphs(int db)` from the class RunScripts.java in the main function of the class Run.java, as bellow:

```
public static void main(String[] args) {

	try {
		//0 is the code of the Dartmouth trace
		RunScripts.runGenerateGraphs(0);

	} catch(Exception e) {
		e.printStackTrace();
		System.out.println(e.getMessage());
	}


}
```

5) Note that the parameter 0 is the code of the Dartmouth trace, since it is the first one in the list bellow, located in the class DB.java.

```
public final static int DB_DARTMOUTH = 0;
public final static int DB_CAMBRIDGE = 1;
public final static int DB_TAXI = 2;
public final static int DB_INFOCOM = 3;
public final static int DB_USC = 4;
public final static int DB_NUS = 5;
public final static int DB_UPB = 6;
public final static int DB_SASSY = 7;
public final static int DB_4SQUARE_NY = 8;
public final static int DB_4SQUARE_TK = 9;
public final static int DB_4SQUARE_SP = 10;
```

6) After you run the code from the Run.java class, three types of graphs will be generated and put on the folder ../Graphs_Data/TraceDB/. The AggregatedX-T.graph contains all the information aggregated from time step 1 to time step T. The WindowedX-T.graph contains only the information within the time step T. The DynamicX-T.graph contains only the active encounters seen at the end of time step T. X is the time step size according to the scale given to the trace. You can check this information in the class DB.java, as bellow:

```
public static int[] DB_RECORD_INTERVAL = {
       3600*24,
       3600*24,
       24,
       1,
       3600*24,
       1,
       3600*24,
       3600*24,
       3600*24*7,
       3600*24*7,
       3600*24*7};

public static int[] DB_WEIGHT_SCALE = {
       3600,
       3600,
       1,
       1,
       3600,
       1,
       3600,
       3600,
       3600,
       3600,
       3600};
```

Tutorial on Recast: generating graphs from new traces

If you have a new trace and want to generate graphs that Recast can read, the only thing you have to do is to create a new scheduler for you, that can read the trace and generate the events. To do this, follow the steps bellow.

1) Go to the package "scheduler" and create a class that extends Scheduler, e.g. SchedulerSassy.java.

2) On your class, the only thing you have to worry is to create the constructor and a method named `public void loadContactTrace() throws IOException`

3) In this method, you have to call the super methods **addConnection** and **addDisconnection** as you read your trace file, considering the duration of the connection (encounter) between persons id1 and id2.

4) After this, you have to fill the DB.java class accordingly. Check, for instance, the method bellow:

```
	public static Scheduler getScheduler() throws Exception {
		if(RunConfig.RUN_OVER_TEST_SET)
			return new SchedulerTestSet(DB.getFilePath());
		else if(myDB == DB_DARTMOUTH || myDB == DB_CAMBRIDGE || myDB == DB_USC ||
myDB == DB_4SQUARE_NY || myDB == DB_4SQUARE_TK || myDB == DB_4SQUARE_SP) {
			return new SchedulerOneFile(DB.getFilePath());
		}
		else if(myDB == DB_TAXI || myDB == DB_INFOCOM) {
			return new SchedulerSeveralFiles(DB.getFilePath());
		}
		else if(myDB == DB_NUS)
			return new SchedulerNus(DB.getFilePath());
		else if(myDB == DB_UPB)
			return new SchedulerUPB(DB.getFilePath());
		else if(myDB == DB_SASSY)
			return new SchedulerSassy(DB.getFilePath());
		return null;
	}
```

5) If the trace file is too big to fill in memory, use the SchedulerOneFile.java strategy. This Scheduler is used to load the Dartmouth trace file. It loads 50000 lines from the trace file into a buffer, then it generates the events from this buffer and, when the buffer is bellow 1/10 of its size, it loads another 50000 lines.

Tutorial on Recast: classifying edges

Once the graphs are created, we can run Recast on them to classify the edges. To do this, we have to execute the main.py file as bellow

```
db = DB.DB()

db.set_db_id(1)

#have to do this once per db
gen_random_graphs(db, 5)

runRecast(db, [0.1, 0.01, 0.001, 0.0001])
```

First, we create a DB class and then we set the DB to 1 (the code for the Dartmouth trace). Then, we say that we want to generate 5 random graphs from the graphs we just generated in the previous step. You can leave the code there, since it will check if you already have generated the random graphs. Once the random graphs are created, you can run Recast to classify the edges using the command "runRecast". The first parameter is is db which you want to run Recast on and the second is a list of P_rnd values that you want Recast to use it. Be sure to have to same number of random graphs as stated in the variable `num_random_files` of DB.py.

After you run, a folder "Graphs_Data/Trace_Label/classified/" will be created with all the classes. There are several files, but the most basic one, with the classes created by Recast is

classifiedDart_class0_threshold0.0001_time30

if your DB is Dartmouth, your p_rand value is 0.0001 and your final time, ie, the time step you use to classify the edges is 30. Basically, the edge class is given in the final column of each row and the nodes involved are the first two columns. Thus, a row like

```
0 149 0.04 0.0 0.16 18.0 43.0 87.0 1 4
```

means that nodes 0 and 140 share the class 4, where class 1 = friends, 2 = bridges, 3 = acquaintances and 4 = random.