BISON
IST-2001-38923

Biology-Inspired techniques for
Self Organization in dynamic Networks

Readme for immune search package (peersim based implementation)

Classification:	Public
Contact Author:	Niloy Ganguly
email:	niloy@zhr.tu-dresden.de
Document Version:	1.0 (February 25, 2005)

Project funded by the
European Commission under the
Information Society Technologies
Programme of the 5^{th} Framework
(1998-2002)

Contents

1 Implementation details - peersim-based isearch package

The implementation of the search algorithm can be roughly divided into the following steps.

- Initialization of the *p2p* network.
- Determination of the desired experiment (either coverage or time-step). This includes both the output as well as the exit condition.
- The algorithm(s) to be executed.

The peersim simulation environment comprises of cycles where at each cycle *protocols* are run and *observers* collect information after every cycle. Moreover, there are *initializers* to initiate the protocols. The protocols, initializers and observers are used to simulate the above mentioned steps of the search algorithm. The *initializers* are used to instantiate the *p2p* network. [Note, here we are reporting only the implementation of simple data/query types.] Since peersim is cycle based and runs for a specified number of cycles, the experiments (coverage or time-step) are not distinguished here. Rather, coverage is run for an arbitrary large number of time steps, so that it is ensured that all nodes are covered. The *observer* consequently outputs the same output for both experiments. Later the output is further analyzed in an excel worksheet to derive the final results.

We outline the implementation of *initializers*, *protocols* and *observers* and also the basic data structure. However, to provide the idea, how our code interacts with the peersim core code, we at first explain the input peersim configuration files which are customized to suit the purposes of the immune search scheme.

1.1 Configuration file

To run a search algorithm, a suitable peersim configuration file is needed. In the following we present the configuration file and go on explaining the inputs. (Note : if a line starts with #, it is a comment line)

```
# PEERSIM EXAMPLE iSEARCH
random.seed 1264967890
simulation.cycles 21
simulation.shuffle

overlay.size 10000

protocol.0 example.isearch.RProliProtocol
protocol.0.ttl 50
protocol.0.walkers 10

init.0 peersim.dynamics.AnyTopology
init.0.protocol 0
init.0.filename TestP4_sorted
```

```
15
16 init.1 example.isearch.SearchDataInitializer
17 init.1.protocol 0
18 init.1.max_queries 100
19 init.1.data_zipf 1.0
20 init.1.query_zipf 1.0
21 init.1.keywordsWidth 10
22
23 observer.0 example.isearch.SearchObserver
24 observer.0.protocol 0
25 observer.0.verbosity 0
```

Line 2 to 6 - Global peersim configuration
Lines from 2 to 6 regard the global peersim configuration, such as the seed for the random number generator, the number of simulation cycles to perform, the node shuffle switch (to avoid picking nodes in the same order at each cycle) and the network size. Line 3 showing the number of simulation cycles to be performed is not needed for running the search algorithm, it is just a part of the historical code of peersim.

Line 8 to 10 - Protocol
The protocol actually represents different algorithms (random walk, proliferation etc.) we run. Here, in the configuration file, we define the random walk protocol (**lines 8 to 10**). The `ttl` parameter defines the number of time steps a particular search item will run. Since, in peersim, the two experiments, *time-step* and *coverage*, are not separately executed; here for running the experiment coverage, the ttl is to be set to an arbitrary large value, so that all the peers are ultimately covered. The `walkers` define the number of message packages initially used to start an experiment.

Line 12 to 21 - Initializers
The first initializer (**lines from 12 to 14**) defines the topology. The topology can be simply read from a file containing information of connections between nodes. The topology file is named in the 14^{th} line.

The second initializer (**lines from 16 to 21**) defines the way key repository and the query distribution are to be initialized at each node. The parameters are **max_queries** - the number of query operations performed for experiment; **data_zipf** and **query_zipf** - the Zipf's coefficient for data and query, respectively; **keywordsWidth** - the width of the token used for query. (Since we are working with simple data sets, each data or token is k bits wide and we have 2^k unique tokens.)

Line 23 to 25 - Observers
Finally, the observer is defined from **line 24 to 26**. The observers output the result at the end of search of each individual cycle or at the completion of a single search operation (defined by verbosity - line 25).

The protocol, initializers and observers are explained one by one.

1.2 Protocol

As mentioned protocols essentially implement the algorithms - random walk and proliferation, both their normal and restricted version. All the protocols have some common basic services. An abstract class `SearchProtocol` provides a first common implementation of those services; all `isearch` protocols are supposed to inherit from this class.

Usually, each protocol is made of two distinct parts: *active* and *passive*; the former represents the pro-active behavior (e.g.: nodes inject queries into the system according to a predefined distribution), and the latter reacts to the incoming messages according to the specific protocol behavior. *send* and *forward* are the common metaphors used to provide these behaviors. The *match* function returns the amount of match between the contents of the node and the content of the incoming token. Each node therefore performs the work of *send* and *forward* at each cycle through the function *nextcycle*.

Individual protocols on top of the framework

Each distinct protocol, random walk (RW), restricted random walk (RRW), proliferation ($Proli$) and restricted random walk ($RProli$), extends the basic `SearchProtocol` and customizes the program according to its underlying guiding algorithm. An example of restricted random walk will clarify this.

A restricted random walk differs from the simple random walk only in the strategy with which a neighbor is chosen. The node neighbors are probed to find a candidate which has never seen before the message that is going to be sent. If there isn't such a candidate, then a random node is chosen as in the random walk protocol. An outline of the RRWprotocol class which extends the standard version (RWProtocol) is given below.

```
public class RRWProtocol extends RWProtocol {
    public RRWProtocol(String prefix, Object obj)  {
        super(prefix, obj);
    }

    public void process(SMessage mes) {
        // checks for hits and notifies originator if any:
        boolean match = this.match(mes.payload);
        if (match) this.notifyOriginator(mes);

        // forwards the message to a random FREE neighbor:
        Node neighbor = this.selectFreeNeighbor(mes);
        this.forward(neighbor, mes);
    }
}
```

The only difference between random walk and restricted random walk is the way the `process()` method adopts to forward the messages. A specific function (`selectFreeNeighbor()` member function of the basic `SearchProtocol` class) supports the restricted protocol version and chooses a neighbor according to the restricted strategy.

1.3 Initializers

In peersim - the model initializers -the topology and the distributions are attached to a specific protocol, here the search protocol.

1.4 Observers

The generation of the query statistic is made by a peersim observer interface object. At the end of each cycle the observer runs and collects the data about the packets stored at each node and generates statistics. At the end of each cycle or at the end of the whole simulation, the observer generates the following data tuples about query packets:

$$\begin{pmatrix} queryID \\ message\ TTL \\ number\ of\ nodes\ the\ query\ has\ visited \\ number\ of\ successful\ hits \\ total\ number\ of\ message\ packets\ used\ for\ this\ query \end{pmatrix}$$

The observer produces an output at the end of each cycle when verbosity = 1. Below we show a sample output of the 7^{th} and 8^{th} query items while running `Proliprotocol`. The number of time steps (ttl) to be executed is 50.

```
.
.
.
7 46 5415 723 7570
7 47 5718 778 8331
7 48 6001 808 9106
7 49 6286 840 9896
8 0 1 0 1
8 1 2 0 2
8 2 3 0 3
8 3 6 2 6
.
.
.
```

If the verbosity = 0, the output is shown at the end of each search operation, The sample output shows the output produced by the 30^{th} to 34^{th} query items while running `Proliprotocol`. The ttl here is 50.

```
.
```

```
.
.
30 49 38 0 63
31 49 780 56 13243
32 49 161 1 989
33 49 328 7 3475
34 49 240 2 385
.
.
.
```

We now give a brief outline of the data structure used by each node which has enabled us to produce the desired output.

1.5 Basic data structure

To run the above mentioned initializers, observers and protocols on the network, the data structures needed by each node in the infrastructure are the following:

- **messageTable**: hashtable that maps a packet to an integer; it represents the number of times the current node has seen this packet before.
- **hitTable**: hashset that stores the packets for which the current node reports a query hit.
- **incomingQueue**: list that buffers the incoming packets; it is accessed in a FIFO fashion.
- **view**: the current node neighbor list view. It is managed by the linkable interface methods. (Linkable interface method is a facility provided by peersim).
- **keyStorage**: hashmap mapping a key to an integer; the latter represents the frequency of the key.
- **queryDistro**: treeset mapping integers to key array; the former represents the cycle in which the query is scheduled and the latter represents the packet query payload.

1.6 Software package details

The peersim simulation platform is available for download from `http://peersim.sourceforge.org/`. The isearch software should be installed into the `example` directory of the peersim installation. It contains the java files; also various configuration files in subdirectory *powerLaw*, *grid* and *randgraph* Moreover, the file AnyTopology.java should be installed in the peersim/dynamics directory and the file Simulator.java to be installed in peersim/cdsim.

However, one can download the entire package (peersim + search) with all the files pre-arranged from the BISON website. Our software consists of *peersim* configuration files, topology data files, and Java source code files. The source files are located in subdirectories.

To run the config file, assuming the peersim classes presence in the CLASSPATH, just type:

```
java peersim.Simulator <path-to-configfile>/config-isearch.txt
```

where config-isearch.txt is a *peersim* configuration file.
An example command which can be directly run when the package is downloaded

```
java peersim.Simulator example/isearch/randgraph/config-rrw-rproli.txt
```