The program is written in C and is provided as source files. The source files must be compiled to produce the simulation program. We mainly used Intel C compiler (icc, version 16.0.4), but the GNU C compiler (gcc) will also work (although the outcome of a simulation will change due to different optimization to numerical calculations).

Installation (Run the default mode)

By default, the program evolves TRNs under selection for filtering out a short spurious signal, and allows the signal to regulate the effector directly. The program runs on 10 CPU cores (Haswell V3 28 core processor), and takes 1-2 days.

We suggest using a Linux system to facilitate the installation. To run the program in the default mode, follow these steps:

1. Copy all source files (files with suffix .c and .h, and the *makefile*) to one directory.

2. Under the same directory, create a folder and name it **result**. The folder will be used to hold output files.

3. Change directory into the directory that contains the source file. Compile source files using the command

```
    make simulator CC=icc
```

This command will create several files with suffix .o and an executable program named simulator. “CC=icc” compiles the source files with icc. By default, the compiling is done with -O3 for simulation speed, and -fp-model precise -fp-model source to ensure arithmetic operations are accurate and reproducible.

To compile with gcc, change “CC=icc” to “CC=gcc” and comment out -fp-model precise -fp-model source in *makefile*. We have noticed that when compiled with gcc, the simulation produces result different from when compiled with icc, even for the same random number seed. Enabling safe arithmetic options in gcc may solve the problem, but we haven’t tested it.

4. Execute simulator to start. On Linux, this is done with the following command

```
./simulator
```

Output

The simulation will generate several files when it begins to run. The size of some files, e.g. evolutionar_summary_481.txt, will keep increasing. Samples of output files and a description to their content can be found in folder **output_sample**.

Run neutral evolution

Neutral evolution is simulated with one CPU and finishes in minutes. To enable this mode, modify line 33 of netsim.h to

```
#define NEUTRAL 1
```

Then compile the source files and run the simulator.

Make selection condition for signal recognition

The selection condition is specified in main.c. By default, the program selects for filtering out a short spurious signal. To create selection for signal recognition, modify line 118 – 127 of main.c to

```
selection.env1.signal_on_strength=1000.0;
selection.env1.signal_off_strength=0.0;
selection.env2.signal_on_strength=1000.0;
selection.env2.signal_off_strength=0.0;
selection.env1.signal_on_aft_burn_in=1;
selection.env2.signal_on_aft_burn_in=1;
selection.env1.t_signal_on=200.0;
selection.env1.t_signal_off=0.0;
selection.env2.t_signal_on=10.0;
selection.env2.t_signal_off=200.0;
```

If the signal is not allowed to directly regulate the effector (see Additional settings), a burn-in condition of evolution is required. To enable burn-in, set line 260 of main.c to

```
burn_in.MAX_STEPS=1000;
```

and line 171 to

```
selection.MAX_STEPS=51000;
```

Also set line 242 – 251 to

```
burn_in.env1.signal_on_strength=1000.0;
burn_in.env1.signal_off_strength=0.0;
burn_in.env2.signal_on_strength=1000.0;
burn_in.env2.signal_off_strength=0.0;
burn_in.env1.signal_on_aft_burn_in=1;
burn_in.env2.signal_on_aft_burn_in=1;
burn_in.env1.t_signal_on=200.0;
burn_in.env1.t_signal_off=0.0;
burn_in.env2.t_signal_on=10.0;
burn_in.env2.t_signal_off=200.0;
```

Output expression levels of genes over time

This mode samples the concentration of proteins over time. It uses the accepted_mutation_x.txt (here x is the random number seed of the simulation. We provide accepted_mutation_481.txt in folder **output_sample** as an example) of a previous simulation to replay evolution, and reproduce the genotype at a given evolutionary step. To enable this mode, following these steps:

1. Modify line 34 of netsim.h to

```
#define PHENOTYPE 1
```

and line 71 of netsim.h to

```
#define SAMPLE_GENE_EXPRESSION 1
```

2. Copy accepted_mutaton_481.txt file (see folder **output_sample**) to result.

3. Modify line 171 of main.c

```
selection.MAX_STEPS=n;
```

The network that evolves at evolutionary step n will be reproduced.

4. Compile the source code and run the simulator

This mode can be run on one or multiple CPUs and finishes in minutes.

The program simulation gene expression under environment A and B, and samples instantaneous fitness and protein concentrations during the simulation. The sampling interval is 1 minute in developmental time. See **readme_output.pdf** in folder **output_sample** for the output files.

Sample parameters of evolved newtwork motifs

We can study the constaint to network motif parameters by sampling parameters from random network motifs. To do this,

1. Modify line 34 of netsim.h to

```
#define PHENOTYPE 1
```

and line 62 of netsim.h to

```
#define SAMPLE_PARAMETERS 1
```

2. Additional settings about sampling are line 63-65 of netsim.h. The code can only sample from one type of network motifs at a time. Which network motifs to sample from is determined by the value of TARGET_MOTIF.

```
#define SAMPLE_SIZE 100 //number of samples to take
#define START_STEP_OF_SAMPLING 41001 //sample from the gen
otypes at the start step and afterwards
```

```
#define TARGET_MOTIF 2 // 0 means sampling genes regardles
s of motifs
                       // 1 samples from c1-FFLs under dir
ection regulation
                       // 2 samples from isolated AND-gate
d C1-FFLs
                       // 3 samples from isolated AND-gate
d FFL-in-diamonds
```

3. Modify line 171 of main.c, so that it is sufficiently larger than the value of START_STEP_OF_SAMPLING in step 2.

```
selection.MAX_STEPS=51000;
```

Based on the settings at step 2 and 3, we will be resampling 100 times for the parameters of an isolated AND-gated C1-FFL from evolutionary step 41001 to 51000.

4. Copy accepted_mutaton_481.txt file (see folder **output_sample**) to result.

5. Compile the source code and run the simulator

Run perturbation analysis

In this mode, the program replays mutation and perturbs TRNs at the given evolutionary steps. The program will exclude a TRN from

perturbation if the topology of TRN confounds the desired modification (e.g. besides the desired motif, another motif is also modified by the perturbation). If a TRN si suitable for perturbation, the program calculates the fitness before and after the perturbation. To enable the perturbation mode, following these steps:

1. Modify line 35 of netsim.h to

```
#define PERTURB 1
```

2. Specify the type of perturbation in line 78 – 84 of netsim.h.

Example 1: For evolutionary step 41001 and afterwards, converting AND-gated isolated C1-FFLs to fast-TF-controlled isolated C1-FFLs by adding a strong binding site

```
#define START_STEP_OF_PERTURBATION 41001
#define WHICH_MOTIF 0 //only one type of motif can be dist
urbed at a time: 0 for C1-FFL, 1 for FFL-in-diamond, 2 for
 diamond
#define WHICH_CIS_TARGET 0 //0 for effector gene, 1 for fa
st TF gene, 2 for slow TF gene
#define WHICH_TRANS_TARGET 1 //0 for signal, 1 for fast TF
, 2 for slow TF
#define ADD_TFBS 1 // 1 for adding a TFBS of the trans tar
get to the regulatory sequence of the cis target,
                   // 0 for removing ALL TFBSs of the tran
s target from the cis target
```

```
#define ADD_STRONG_TFBS 1 //by default, we add TFBSs with
high binding affinity to change topology and/or logic
```

Example 2: For evolutionary step 41001 and afterwards, convert AND-gated FFL-in-diamonds to AND-gated isolated diamonds

```
#define START_STEP_OF_PERTURBATION 41001
#define WHICH_MOTIF 1 //only one type of motif can be dist
urbed at a time: 0 for C1-FFL, 1 for FFL-in-diamond, 2 for
 diamond
#define WHICH_CIS_TARGET 2 //0 for effector gene, 1 for fa
st TF gene, 2 for slow TF gene
#define WHICH_TRANS_TARGET 1 //0 for signal, 1 for fast TF
, 2 for slow TF
#define ADD_TFBS 0 // 1 for adding a TFBS of the trans tar
get to the regulatory sequence of the cis target,
                    // 0 for removing ALL TFBSs of the tran
s target from the cis target
#define ADD_STRONG_TFBS 1 //by default, we add TFBSs with
high binding affinity to change topology and/or logic
```

3. Modify line 171 of main.c to specify the last evolutionary step to be perturbed.

4. Copy *accepted_mutation_x.txt* file and *evo_summary_x.txt* to result.

5. Compile the source files and run simulator.

Because the program needs to measure the fitness of many TRNs, it is recommended to run the program with multiple CPUs. See **readme_output.pdf** in folder **output_sample** for the output files.

Additional settings

1. Change random number seed

Random number seed is set at line 37 of main.c. It mainly controls the initial genotypes.

2. Change the number of parallel threads

By default, the program runs on 10 threads. To change, modify line 41 of netsim.h. Note that N_REPLICATES (line 42 of netsim.h) must be divisible by N_THREADS!

3. Change output interval

By default, the program pools results of 20 evolutionary steps before writing to disk. This can be changed by modifying OUTPUT_INTERVAL at line 44 of netsim.h.

4. Direct regulation of signal to effector

By default, the program allows the signal to evolve to directly regulate the effector. To disable this, change line 55 of netsim.h to 1. Evolutionary burn-in is recommended if direct regulation is not allowed.

5. Penalty of undesirable effector

By default, the effector is harmful if expressed in a wrong environment. To remove the harm (the cost of expressing the effector still applies), set 162 of main.c to l (harm”l”ess).

6. Count near-AND-gated motifs

By default, near-AND-gated motifs are not counted. Set line 93 of netsim.h to count them.

7. Excluding weak TFBSs when scoring motifs

By default, TFBSs with up to 2 mismatches are included when scoring motifs. Line 94 - 97 of netsim.h set the maximum number of mismatches in a TFBS.