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).

We provide two versions of the source files: the original, which was used to generate all results in the manuscript, and the revised, which is restructured to make it easier to understand. When a high optimization level (-O3) is enabled during compilation, the two versions produce different simulation results when the same random number seed is used. The difference is caused by unsafe optimization to numerical computations during compilation, therefore should only change the quantitative results of individual simulation replicates, but not the conclusion of our research. When safe optimization options (-fp-model source -fp-model precise) are turned on, the two versions produce identical results. Users who want to reproduce the exact results in our paper can find the original version in branch **manuscript-version**. The original version comes with its own readme and a list of random number seeds we used. This readme aims to explain the revised version.

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 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; change it to “CC=gcc” to compile with gcc. To minimize the impact of hardware difference on results, modify line 2 of the *makefile* to

```
CFLAGS = -O3 -fp-model source -fp-model precise
```

if icc compiler is used.

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_5.txt, will keep increasing. Samples of output files and a description to their content can be found in folder **output_sample**.

The following settings are different for the original version of the code; please refer to different readme to set the original code.

Run neutral evolution

Neutral evolution is simulated with one CPU and finishes in minutes. To enable this mode, modify line 21 of netsim.h to

```
#define NEUTRAL 1
```

Then compile the source file 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 98 – 105 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.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 225 of main.c to

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

and line 152 to

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

Also set line 209 – 216 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.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_mutations_x.txt file 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 22 of netsim.h to

```
#define PHENOTYPE 1
```

2. Copy accepted_mutatons_5.txt file (see folder **output_sample**) to result.

3. Modify line 152 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 runs replicate of developmental simulation under environment A and B, and samples instantaneous fitness and protein concentrations during simulation. The default sampling interval is 1 minute in developmental time. See **readme_output.pdf** in folder **output_sample** for the output files.

Run perturbation analysis

In this mode, the program replays mutation and attempt perturbation on TRNs at the given evolutionary steps. The program will exclude a TRN if it is not suitable for the perturbation. If a TRN can be perturbed, the program calculates the fitness before and after the perturbation. To enable the perturbation mode, following these steps:

1. Modify line 23 of netsim.h to

```
#define PERTURB 1
```

2. Specify the type of perturbation in line 52 – 59 of netsim.h.

Example 1: converting AND-gated isolated C1-FFLs to fast-TF-controlled isolated C1-FFLs by adding a strong binding site

```
#define DISABLE_AND_GATE 1
#if DISABLE_AND_GATE
#define WHICH_MOTIF 0
#define ADD_STRONG_TFBS 1
#define FORCE_MASTER_CONTROLLED 1
#endif
#define FORCE_DIAMOND 0
#define FORCE_SINGLE_FFL 0
```

Example 2: convert AND-gated FFL-in-diamonds to AND-gated isolated diamonds

```
#define DISABLE_AND_GATE 0
#if DISABLE_AND_GATE
#define WHICH_MOTIF 0
#define ADD_STRONG_TFBS 0
#define FORCE_MASTER_CONTROLLED 0
#endif
#define FORCE_DIAMOND 1
#define FORCE_SINGLE_FFL 0
```

3. Copy *accepted_mutations_x.txt* file and *evo_summary_x.txt* to result.

4. By default, the program tries to perturb TRNs at the last 10,000 evolutionary steps. Line 152 of main.c specifies the last evolutionary step to modify, and line 399 of netsim.c specifies the number of evolutionary steps to modify.

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 26 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 29 of netsim.h. Note that N_REPLICATES (line 30 of netsim.h) must be divisible by N_THREADS!

3. 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 44 of netsim.h to 1. Burn-in is recommended if direct regulation is not allowed.

4. 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 line 45 of netsim.h to 1

5. Count near-AND-gated motifs

By default, near-AND-gated motifs are not counted. Set line 65 of netsim.h to count them. Note that counting near-AND-gated motifs is available only when DIRECT_REG = 0 and PHENOTYPE = 1.

6. Count long-arm C1-FFLs

By default, long-arm C1-FFLs are not counted. Set line 66 of netsim.h to count them. Note that counting long-arm C1-FFLs is available only when DIRECT_REG = 1.

7. Excluding weak TFBSs when scoring motifs

By default, TFBSs with up to 2 mismatches are included when scoring motifs. Line 69 - 72 of netsim.h set the maximum number of mismatches in a TFBS.