Samples of the output files are provided in folder *output_sample*. They can be reproduced by compiling the source code with icc and running simulation on Intel Haswell V3 28 core processor or Intel Xeon E5-2660 V2 20 core processor.

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 CPUs, and takes 1-2 days.
To run the program in the default mode, follow these steps:

1. Download all source files 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. Compile source files using command
```
make simulator CC=icc
```

4. Execute *simulator* to start simulation. On UNIX systems, this is done with the following command
```
./simulator
```

Output

The output files have the same format as the those produced by the revised code do.

Run neutral evolution

Neutral evolution is simulated with one CPU, and finishes in minutes. To enable this mode, modify line 16 of netsim.h to

```
#define NEUTRAL 1
```

and modify line 17 to

```
#define RUN_FULL_SIMULATION 0
```

Then compile the source file and run *simulator.*

Make selection condition for signal recognition

The selection condition is specified in netsim.c. By default, the program selects for filtering out a short spurious signal. To create selection for signal recognition, modify both line 3473 – 3478 and line 3965 - 3970 of netsim.h to

```
env1_signal_strength=1000.0;
env2_signal_strength=0.0;
env1_t_signal_on=200.0;
env1_t_signal_off=0.0;
env2_t_signal_on=100.0;
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 23 of netsim.h to

```
    #define BURN_IN 1000
```

and line 22 to

```
    #define MAX_MUT_STEP 51000
```

Setting the burn-in condition has three steps.

1. The fitness of the initial genotype needs to be calculated under the burn-in condition; the weight of the two environments needs to be reversed. To do this, change line 3976 and 3977 to

```
    env1_occurence=0.67;
    env2_occurence=0.33;
```

2. By default, line 3358 – 3363 are the burn-in condition of selection for spurious signal filter. Modify the lines to

```
    env1_signal_strength=1000.0;
    env2_signal_strength=1000.0;
    env1_t_signal_on=200.0;
    env1_t_signal_off=0.0;
    env2_t_signal_on=10.0;
    env2_t_signal_off=200.0;
```

3. By default, line 3413 - 3418 are the selection condition for spurious signal filter. Modify the lines to

```
    env1_signal_strength=1000.0;
    env2_signal_strength=1000.0;
    env1_t_signal_on=200.0;
    env1_t_signal_off=0.0;
    env2_t_signal_on=10.0;
    env2_t_signal_off=200.0;
```

Plot expression of genes over time

This mode samples the concentration of proteins over time. It uses the accepted_mutation_x.txt file of a previous simulation to replay evolution, and reproduce the genotype at a given evolutionary step. To enabled this mode, following these steps:

1. Modify line 13 of netsim.h to

```
    #define PHENOTYPE 1
```

2. Copy an *accepted_mutatons_x.txt* file to *result*. Make sure the value of “x” is the same as the random number seed specified in main.c (line 17).

3. Modify line 22 of netsim.h to

```
    #define MAX_MUT_STEP n
```

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

4. Compile the source code and run *simulator*
This mode can be run on one or multiple CPUs, and finishes in minutes. The program runs replicates 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. The sampled instantaneous fitness is stored in *fitnessA.txt* and *fitnessB.txt*. By default, *simulator* only output the concentration of protein, which is the sum of all variants of a protein, is stored in *protein_A.txt* and *proteinx_B.txt*. To output the concentration of a protein variant, set line 14 of netsim.h to 0.

Run perturbation analysis

In this mode, the program also replay 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 15 of netsim.h to

```
    #define PERTURB 0
```

2. Specify the type of perturbation in line 57 – 64 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 FORCE_NON_AND_GATE 1
    #if FORCE_NON_AND_GATE
    #define WHICH_MOTIF 0
    #define ADD_STRONG_TFBS 1
    #define FORCE_FAST_TF_CONTROLLED 1
    #endif
    #define FORCE_DIAMOND 0
    #define FORCE_ISOLATED_FFL 0
```

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

```
    #define FORCE_NON_AND_GATE 0
    #if FORCE_NON_AND_GATE
    #define WHICH_MOTIF 0
    #define ADD_STRONG_TFBS 0
    #define FORCE_FAST_TF_CONTROLLED 0
    #endif
    #define FORCE_DIAMOND 1
    #define FORCE_ISOLATED_FFL 0
    #endif
```

3. Copy an *accepted_mutations_x.txt* file and *an evo_summary_x.txt* to *result*. Make sure the value of "x" is the same as the random number seed specified in main.c.

4. By default, the program tries to perturb TRNs at the last 10,000 evolutionary steps. Line 22 of netsim.h specifies the last evolutionary step to modify, and line 3244 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.

Additional settings

1. Change the number of parallel threads
By default, the program runs on 10 threads. To change, modify line 26 of netsim.h. For example,

```
#define N_TREADS 5
```

sets the number of threads to 5. Note that N_REPLICATES (line 27 of netsim.h) must be divisible by N_THREADS.

2. 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

```
#define DIRECT_REG 0
```

Burn-in evolution is required to if the signal is not allowed to regulate the effector directly.

3. 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), modify line 45 of netsim.h to

```
#define NO_PENALTY 1
```

4. Output error log
By default, the program keeps a short log of errors. To log more errors, which are mostly numerical errors, set line 41 of netsim.h to 1.

5. Count near-AND-gated motifs
By default, the program does not count near-AND-gated motifs. To count them, set line 65 of netsim.h to 1. We currently count near-AND-gated motifs only when the signal cannot directly regulate the effector.

6. Count C1-FFLs with a long arm
By default, the program does not count C1-FFLs with a long arm. To count them, set line 66 of netsim.h to 1. We currently count near-AND-gated motifs only when the signal can directly regulate the effector.

7. Excluding weak TFBSs when scoring motifs
By default, the program counts in weak TFBSs when scoring motifs. To exclude weak TFBSs, modify line 67 – 70 of netsim.h.