Visual GEC Tutorial

Background

Visual GEC is a programming language for designing, simulating and analysing genetic devices that can be inserted into cells to reprogram their behaviour. The tool is intended for use by researchers and students with an understanding of basic concepts in synthetic biology.

Preliminary: Installing Visual GEC

1. Navigate to http://research.microsoft.com/gec and click on the "Web Simulator" link.
2. Visual GEC requires Silverlight 4 to be installed on your machine. If this is not the case, you should receive a notification in your browser together with installation instructions. Silverlight 4 is available for Windows and Mac as a plugin for most major web browsers.
3. In the top-right corner, click on the "Install" button. This will install the software to your local machine for offline use. You will be presented with options to create shortcuts on the Start menu and the Desktop. If you have already installed the tool, the "Install" button will read "Update" and will check if a newer version of the tool is available to download.

I: Basic compilation and simulation

1. Load the "Basic" example from the "GEC Examples:" drop-down menu. The program code will appear in the "GEC" tab on the left-hand side. The code describes a sequence of genetic parts which, when inserted into a cell, will be read by the cell machinery to produce proteins that behave in specific ways. Visual GEC contains a database of genetic parts which it uses to automatically construct systems which satisfy specific design criteria. Inspect the database by navigating to the "Database" tab on the left-hand side. The four part types are protein coding regions (pcr), promoters (prom), ribosome binding sites (rbs) and terminators (ter). They each have associated properties, e.g. the property "codes(xylR, 0.001)" for the protein coding region "i723017" specifies that the part codes for the protein xylR, and that this protein has an intrinsic degradation rate of 0.001.
2. Navigate back to the "GEC" tab. The example implements a negative feedback loop by coding for a protein Y which negatively regulates its own production via an associated promoter. The use of a variable Y to stand for the protein indicates that we do not care which protein is used. Thus there may be multiple ways to select parts from the database to build a system with this behaviour. Click on the "Solve GEC" button to automatically generate all possible system designs. The solutions are presented in a drop-down "SOLUTIONS:" menu below the code editor window in the "GEC" tab. The solutions are determined by selecting all possible combinations of parts from the database that satisfy the design constraints. How many solutions are there?
3. Select a numbered solution from the drop-down menu of solutions (other than the Solution template). Notice that the protein assigned to the variable Y is displayed below the drop-down menu, in addition to the corresponding set of genetic parts for constructing this system. Now switch to the "LBS" tab on the left-hand side. Selecting a solution populated

the “LBS” tab with program code written in the Language for Biochemical Systems (LBS), developed by Michael Pedersen and Gordon Plotkin (LNCS 5307:63-82, 2008). The LBS code is a low-level program consisting of the full set of chemical reactions that correspond to the chosen solution. These reactions include transcription and translation, as well as degradation of mRNA and proteins.

4. Use the “Simulate LBS” button to run a stochastic simulation of the LBS code, observing how the population of the protein (blue) varies with time. The negative feedback should produce sharp spikes in the population of the protein.
5. Go back to the “GEC” tab and select different numbered solutions, and simulate those as well for comparison (you don’t need to go back to the LBS tab for each solution: just select the solution from the “SOLUTIONS:” menu and click “Simulate LBS”). The behaviour of the various solutions is slightly different because the different proteins have different transcription, translation, degradation and negative regulation rates. Which solution gives the cleanest spikes?
6. Visual GEC also includes a deterministic simulator which constructs and solves an ODE representation of the system. Select “Deterministic” from the “Simulation:” drop-down menu, then re-run the LBS simulation for the best solution. Is the deterministic simulation helpful in this case?
7. As new genetic parts are characterised they can be added to the database, thereby increasing the range of systems that can be constructed using the Visual GEC tool. Go to the “Database” tab and click on the “Add” button to add a new line in the parts database. Select the new line and invent an ID for the new part (which is different from the existing identifiers in the database). Fill in the part type and properties fields to produce a protein coding region which codes for araC with degradation rate 0.01. Finally, check the box in the “Enabled” column. This part is now ready for use in the construction of new genetic devices.
8. Re-solve the GEC program. There should be an additional solution which uses the newly-added part. Select it and run a stochastic simulation. How do the spikes compare to the other solutions? The degradation rate of the protein is an important factor in the behaviour of the system. Try modifying the degradation rate for the new part in the database, then re-solve the GEC program and simulate the solution which uses the new part. What happens if the degradation rate is very low?

II: A more complex example

1. Reset the parts database by clicking the “Reset” button in the “Database” tab, then load the “Repressilator” example program from the “GEC Examples:” drop-down menu. This example describes three proteins that form a negative feedback cycle, which produces a genetic oscillator in which the populations of the three proteins alternate between low and high in turn (Nature 403:335-338, 2000).
2. Solve the GEC program and compare stochastic simulations of solutions 1, 3 and 7. Which produces the best oscillations? Pick a few more solutions at random and test to see whether they oscillate.
3. In many of the solutions, one protein binds more tightly to its corresponding promoter region than the other proteins, allowing a single protein to dominate. Load the “Repressilator Similar” example program. This example extends the previous example with

some additional constraints that the unbinding rates for all three proteins should be roughly similar. Solve this GEC program and observe that the number of solutions has been reduced. Run simulations for each of the solutions – how many produce oscillatory behaviour?

III: Modular programming

1. Load the "Repressilator Modules" example program. This is another version of the Repressilator example in which the repeated design motif has been encapsulated in a module definition, which is then instantiated three times with different parameters. The "gate(A,B)" declaration produces a gate which is negatively regulated by A and produces the output protein B. Two more declarations complete the negative feedback loop as before. Solve this program and check that it produces the same number of solutions as the original "Repressilator" example.
2. This gate component can also be used to produce a bistable switch instead of an oscillator. This system requires two gates separated by a vertical bar (for parallel composition): the first is negatively regulated by A and has output B whereas the second is negatively regulated by B and has output A. Modify the last line of the example program to implement a bistable switch as described above. How many solutions are there for the GEC program? Run multiple stochastic simulations for a few of the solutions to verify that they are in fact bistable. Use the "Pause" button to abort the current simulation and then use "Simulate LBS" to start a new simulation. Each simulation run is different, due to the stochastic nature of the simulation.

Answers

Part I, number 2: With the default database there are 4 solutions.

Part I, number 5: Solution 1 gives the best spikes, separated by periods where the protein concentration is almost zero.

Part I, number 6: The deterministic simulation is not helpful because the spikes are produced by stochastic effects, hence the deterministic simulation does not show the same behaviour.

Part I, number 8: The type should be “pcr” and the property should be “codes(araC, 0.01)”. The corresponding solution to the GEC program should be number 2, and in the simulation the spikes are particularly short. If the degradation rate is very slow then the spikes remain for much longer.

Part II, number 2: Solution 7 gives the best oscillations.

Part II, number 3: The additional constraints in this example mean that all solutions produce the desired oscillatory behaviour.

Part III, number 2: the last line should become “gate(A,B) | gate(B,A)”.