

# Computational evolution of gene circuit topologies to meet design requirements

Lewis Grozinger<sup>1</sup>, Ángel Goñi-Moreno<sup>1</sup>

<sup>1</sup>Centro de Biotecnología y Genómica de Plantas, Universidad Politécnica de Madrid (UPM)-  
Instituto Nacional de Investigación y Tecnología Agraria y Alimentaria (INIA/CSIC), Madrid, Spain  
lewis.grozinger@upm.es / angel.goni@upm.es

## Abstract

The design and implementation of synthetic gene regulatory networks that compute is a central effort to synthetic biology. Genetic components are arranged into circuits to perform pre-defined functions such as logic gates or bistable switches in living cells. Despite the success of the field there is vast room for improvements since the mechanistic workings of living systems are still largely unknown. For example, the implementation of synthetic circuits often follows topological rules from engineering, with genetic gates arranged one after the other to mimic an electronic circuit. However, natural regulatory networks have evolved architectures full of feedbacks, redundancies and unexpected or counter-intuitive (to us) connections. Here, we computationally explore that search space of topological arrangements for synthetic networks. We fix circuit parameters, define output dynamics, and use an evolutionary algorithm based on Cartesian Genetic Programming to evolve solutions that can be realised as synthetic gene networks. Results suggest there are emergent properties hidden in counter-intuitive implementations that impact decisively on phenotypic responses—an aspect so far neglected. We use stochastic simulations to measure the results against both human designed networks, and use the design of a genetic inverter and the design of a genetic AND gate as example problems.

## Introduction

In cellular computing and synthetic biology (Andrianantoandro et al., 2006; Amos and Goñi-Moreno, 2018; Benenson, 2012) gene regulatory networks are engineered to transform their inputs into outputs according to predefined functions. Most commonly these functions are combinatorial logic that are analogues to electronic circuits performing operations such as NOT, AND or NOR (Wang et al., 2011; Tas et al., 2021a). The principles according to which these gene networks are designed reflects the link to electronic circuits, in that complex networks—often referred to as circuits—are built from simpler networks that are assumed to be modular to each other and orthogonal to other cellular mechanisms- In this context, increasing the complexity of the function performed is simply a matter of scaling the network (Nielsen et al., 2016).

These assumptions are incredibly useful and effective (to some extent, at least) because they make the design process much easier. Breaking network designs into simpler chunks allows us to both develop new understanding of existing cellular processes, and to conceive of designs for new functions by using existing ones as building blocks (Del Vecchio, 2015). In machines, algorithms for digital logic synthesis can use these assumptions to design vast networks very quickly, and these same kinds of algorithms have already been used to layout networks for genetic logic circuits (Appleton et al., 2017).

However, there are many challenges to modularizing and scaling gene networks in this way. For example, genetic components do not perform in isolation but have unknown interactions with their background. These interactions, or dependencies (Tas et al., 2021b; Del Vecchio, 2015), can turn a functional genetic logic gate in one cellular host into non-functional one in another. Moreover, it is often difficult to interpret gene expression events as discrete on/off signals. Indeed, molecular signalling is intrinsically a stochastic event leading to analogue, continuous and noisy information processing (Goñi-Moreno and Amos, 2012; Goñi-Moreno et al., 2017). As a result the digital abstraction is not always valid, and extra effort needs to be done in shaping binary performance (Calles et al., 2019). There are many other challenges in using living matter for computation, from standardization to evolution (Beal et al., 2020; Castle et al., 2021). Here, we focus on an aspect thus far neglected, which is the topology of genetic networks.

The topology of natural genetic networks is not as straightforward as in electronic circuits. For example, the TOL regulatory and metabolic network of the soil bacteria *Pseudomonas putida* (de Las Heras et al., 2012; Kim et al., 2021, 2019) has feedbacks, redundancies, spatial constraints, and what seems to us unnecessary steps. The topology is counter-intuitive, and we would design the same function in a simpler way using less components and cellular resources. But we can be sure that that architecture evolved because it is a good solution to a problem that is yet unknown to us. In contrast to this, synthetic ge-

netic networks do resemble straightforward electronic layouts (Nielsen et al., 2016). Would other, counter-intuitive, topologies perform better? That is the question underpinning this article.

Here, we use evolutionary computing algorithms to forward engineer the topology of a gene network to maximise an predefined objective. To this end, we adapt for our purpose Cartesian genetic programming (CGP) (Miller, 1999), an algorithm for the global optimisation of programs that can be represented with a tree or graph-like structure. CGP belongs to a family of optimisation methods inspired by biological evolution. As a gradient-free method, it can be applied to a wide-variety of objectives, including optimizing the results of stochastic simulations of gene networks such as those used in this work.

CGP has previously been applied in combination with parameter estimation to reverse engineer biochemical networks (Nobile et al., 2013). In the following we adapt CGP to forward engineer a gene network to fit a desired behaviour. Our goal is not to fine-tune components, so we do not optimize kinetic rates, but instead we use CGP to evolve the topology of a network using already characterized genetic components. We compare the efficiency of this evolutionary search against random search for various sizes of the design space, and apply the algorithm to design two gene networks of interest, a genetic inverter and a genetic AND gate. The performance of the gene networks proposed by the evolutionary process are measured using stochastic simulation and benchmarked against human-designed gene networks.

## Methods

### Definition of gene networks as directed graphs

The genetic algorithm operates on directed graphs that are defined as sets of vertices  $V$  and edges  $E$ . Promoters (and their cognate transcription factors) are chosen to be vertices, and  $V$  is the genetic part “library” available to the genetic algorithm. An example of a realisation of such a library that has been used previously to engineer genetic logic gates is in Nielsen et al. (2016), providing the inspiration for the graph formulation used here.

The transcription factors expressed from one promoter bind to and repress transcription from their target promoters, and so edges are directed and represent transcriptional inhibition interactions between the promoters. If  $v_i, v_j \in V$  and there is an edge  $(v_i, v_j) \in E$ , then in the decoded gene network, promoter  $v_i$  controls the expression of the cognate repressor for promoter  $v_j$ .

If required the graph may contain a special set of source vertices  $V_{in}$  that represent the inputs of the gene network. A source vertex  $v_i \in V_{in}$  always appears as the first element of an edge, so that  $v_i \in V_{in}$  implies  $(v_j, v_i) \notin E$  for all  $v_j \in V$ .

There may also be a set of sink vertices  $V_{out}$  for the outputs of the gene network. They have the restriction that  $v_i \in V_{out}$  implies  $(v_i, v_j) \notin E$  for all  $v_j \in V$ .

### Gene network simulation

The solutions proposed by the algorithm are evaluated based on the results of simulations run with stochastic algorithms. In most cases presented Gillespie’s direct method is chosen, but for some of the larger networks the rejection stochastic simulation algorithm (RSSA) is used. Both are exact methods and produce the same results, but RSSA is less computationally expensive for the larger networks.

For each promoter  $v_i$  and its cognate repressor  $r_i$ , controlling the expression of repressor  $r_j$ , the following reactions are modelled:

- The translation of the mRNA of  $r_i$  into an  $r_i$  monomer.
- The degradation of the mRNA of  $r_i$ .
- The reversible dimerisation of  $r_i$  monomers.
- The reversible binding of  $r_i$  dimers to  $v_i$ .
- The transcription, at a low rate, of  $r_j$  mRNA from bound  $v_i$ .
- The transcription, at a higher rate, of  $r_j$  mRNA from unbound  $v_i$ .

The parameters of the above processes are selected based on Sanassy et al. (2014), using parameters that are within biologically reasonable ranges. Simulations also include dilution due to division of the cells at exponentially distributed time points, when division occurs the mRNA and protein are partitioned between the mother and daughter cells according to a Binomial distribution. Doubling occurs on average every 90 minutes.

Simulations produce time-course trajectories for all the chemical species in the network. Many trajectories are produced and sampled at random times to estimate the distribution of abundance of a species from the network.

### Random search

In the random search the design space is sampled using the same code that generates random genomes for the genetic algorithm during the mutation process. This is in order to keep the comparison fair, since the design space does not include *all* directed graphs, only those allowed by the definition of the graphs given above.

Each such randomly generated genome is then scored using the same objective function as used in the genetic algorithm. The search is stopped when the value of this objective function is greater than or equal to the score of a benchmark genome specific to the problem. The speed of convergence is then measured as the number of these objective function evaluations.

## Genetic algorithm

The algorithm used is based on cartesian genetic programming (CGP) and follows the following pseudocode steps.

```
x0 = random_genome()
for i = [1, G]
    x1 = mutate(x0)
    g0 = decode_genome(x0)
    g1 = decode_genome(x1)
    if objective(g1) >= objective(g0)
        x0 = x1
return x0
```

Each iteration of the algorithm is called a generation. Each run has  $G$  generations, each which generate a single solution. A solution defines a directed graph as defined above with vertices  $V$  and edges  $E$ . The solution is actually a list of numbers between 1 and the total number of vertices,  $|V|$ , which is parsed as an edge list defining  $E$  for a given  $V$ . Each edge is directed and can appear in the graph only once (repetitions are ignored). Self-edges such as  $(v_1, v_1) \in E$  are allowed. Mutations occur at a rate  $MR$  (in this study  $MR = \frac{4}{n}$ ,  $n$  defined below), and changes a vertex in the solution to another vertex in  $V$ . In this work crossover is not used.

For each generation  $\lambda$  (in this study  $\lambda = 4$ ) mutant solutions are generated from the current parent solution. The mutants are decoded to produce graphs according to the definition given above and then evaluated against an objective function using the simulation and sampling as described above. If the best performing mutant has a performance greater than or equal to the current parent, this mutant becomes the parent in the next iteration.

The first parent is a randomly generated genome of length  $n$ . The length is chosen depending on the size of the set of vertices in the graph  $|V|$ , and the number of input and output nodes  $|V_{in}|$  and  $|V_{out}|$ , so that all possible graphs can be encoded. After a total of  $G$  generations the current parent is returned as the solution proposed by the algorithm.

## Results

### Inverting gene expression

The simplest known example of a genetic logic gate is the genetic inverter. Genetic inverters are single input and single output gene networks which transform high expression of the input gene to low expression of the output gene, and vice versa. An obvious implementation involves a single promoter controlling the expression of the output gene, and whose activity is inhibited by the product of the input gene, as shown in Figure 1A. This is the gene network that seems reasonable and that a human might design, and simulating this gene network shows that the expression of the output is indeed inverted with respect to the expression of the input. Figure 1B shows an example of two stochastic trajectories,

where the expression of the output in the case of low expression of the input (orange line) is higher than in the case of high expression of the input.

Since the operation of the gene network is stochastic, as are the trajectories of the simulation presented in Figure 1B, it is possible that the expression of the output gene may not faithfully invert the expression of the input gene. If we take two measurements  $a_i$  and  $b_i$ , the first from a genetic inverter which is supposed to have low expression of the output, and the second which is supposed to have high expression of the output, then we should expect  $a_i < b_i$  and  $b_i - a_i > 0$ . In Figure 1C the difference distribution, the distribution of  $z = b - a$ , is constructed by sampling from 1024 stochastic trajectories of the type shown in Figure 1B. The part of the distribution below 0 represents errors in the operation of the genetic inverter. Naturally, the goal of an engineer is to minimise these errors.

To express this goal in a form which can be maximised by the genetic algorithm we choose to calculate the signal-to-noise ratio (SNR) of the samples in the distribution of Figure 1C. Given samples  $z_i = b_i - a_i$  for  $i \in [1, n]$  the SNR is computed as  $SNR = \frac{\bar{Z}}{S_Z}$ , where  $\bar{Z} = \frac{1}{n} \sum_{i=1}^n z_i$  is the sample mean and  $S_Z = \frac{1}{n-1} \sum_{i=1}^n (z_i - \bar{Z})^2$  is the sample standard deviation. Calculated in this way, the human designed genetic inverter has an  $SNR = 1.53 \pm 0.03$ . Increasing the SNR means increasing the mean of the difference distribution, and/or decreasing the standard deviation, both of which will improve the reliability of inverter, in terms of reducing its error rate. Intuitively this means shifting the difference distribution to the right and making it narrower.

Using SNR as the objective function the genetic algorithm can evolve an inverter design from a set of 7 genes ( $|V| = 7$ ), and the typical results are shown in Figure 2. Starting with a randomly generated network of the 7 genes in generation 1 (blue line), the difference distribution is broad, has a relatively low mean, and includes a significant proportion of errors (the part of the difference distribution below 0). This particular difference distribution has  $SNR = 0.46 \pm 0.03$ . With each generation the network improves, squeezing the difference distribution and shifting it to the right, until at generation 32 (red line) the difference distribution closely resembles that of the human designed benchmark (filled orange). Indeed the distribution at generation 32 has  $SNR = 1.51 \pm 0.03$ , which is within the standard error for the SNR of the benchmark, and is a successful genetic inverter design.

### Comparison with random search

The algorithm produces solutions that perform comparably to the original inverter design. These solutions would have been found *eventually* by random search of the design space. Therefore, in order to demonstrate the advantage of using the genetic algorithm over such a random search, we now turn our attention to how efficiently the algorithm searched

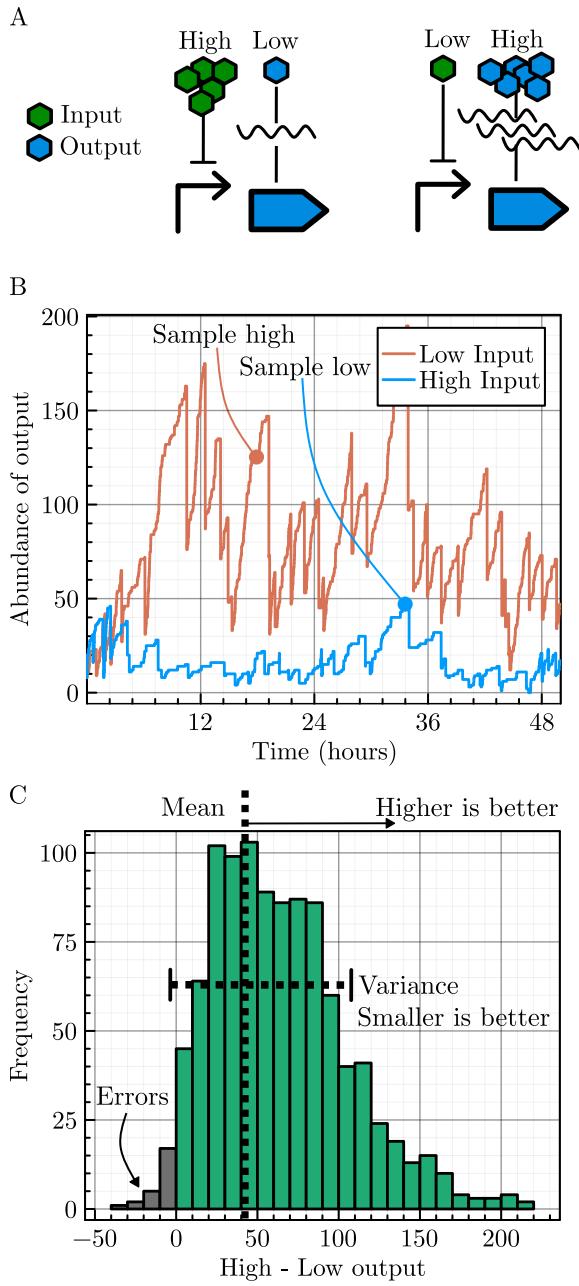


Figure 1: The genetic inverter and scoring its fitness. **A** The intuitive human design of the genetic inverter network. The network is small, containing only the input, which inhibits the expression of the output from the single promoter. If the input is abundant (high) then expression of the output is inhibited and will be less abundant (low), and vice versa. **B** Samples of the abundance of the output gene product (the protein expressed by the output gene) are taken from many different trajectories generated by a stochastic simulation of the inverter. Samples are taken separately for both low (orange) and high (blue) input expression conditions. **C** Samples from low input trajectories are subtracted from samples from those with high input. In an inverter, this should give a distribution with a positive mean. The negative portion of the distribution represents the errors in the operation of the inverter. The goal is to eliminate these errors by increasing the mean of distribution and decreasing the variance.

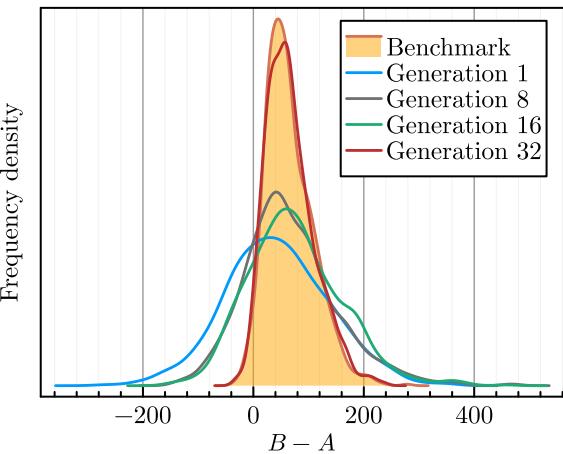


Figure 2: The density of the difference distributions after various numbers of generations of the genetic algorithm, with the goal of evolving a design for a genetic inverter. At generation 1 (blue), the random population performs relatively poorly and manages an  $SNR = 0.46$ . At generation 8 (grey),  $SNR = 0.79$  and at generation 16 (green)  $SNR = 0.88$ . This run of the algorithm was stopped after 32 generations (red), after reaching  $SNR = 1.51$ , which is similar to the human designed benchmark design at  $SNR = 1.53$ .

the design space to find these solutions. Experiments were run to examine the convergence properties of both random search and the algorithmic search in order to compare their efficiency.

To do this the original inverter design was chosen as a benchmark. The number of evaluations of the objective function required before finding a network whose fitness was equal to or greater this benchmark was measured, for both the case of a random search and search guided by the genetic algorithm. The search was performed 64 times for graphs with 3, 4, 5, 6 and 7 vertices ( $|V| \in [3, 4, 5, 6, 7]$ ) to obtain an estimate of the mean number of function evaluations required for each problem size. These estimates and their standard error are shown in Table 1.

At problem sizes of  $|V| = 6$  the genetic algorithm begins to outperform random search. At this point the variance in the number of iterations required also decreases relative to random search, as indicated by the lower standard error for the genetic algorithm in Table 1. Figure 3A shows the full results for these experiments in a box plot, where it can be seen that growth in number of function evaluations required in the genetic algorithm levels off at  $|V| > 6$ , whereas random search continues to increase exponentially. This result is expected for random search, since the size of the search space (number of possible graphs) increases exponentially with  $|V|$ . These experiments confirm that the genetic algo-

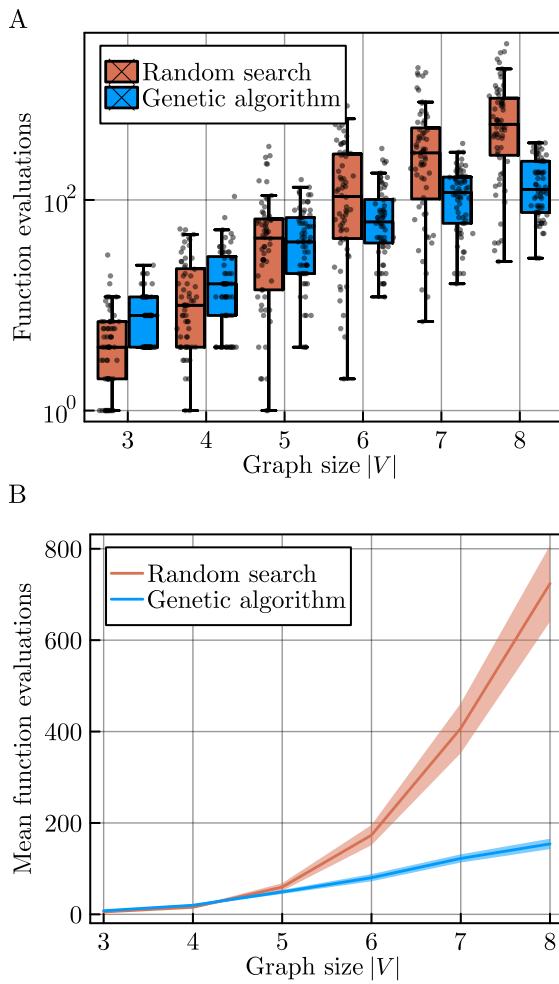


Figure 3: The search efficiency of the genetic algorithm compared to random search. **A** The complete results of 64 searches (each grey dot is a search) at each problem size ( $|V|$ ) from 3 to 8. The searches were run until the fitness of the solution was equal to or greater than that of the benchmark solution, in this case the human-designed genetic inverter. In terms of function evaluations, the genetic algorithm begins to outperform random search at around  $|V| = 6$ . Moreover the variance is efficiency is notably lower for the genetic algorithm. **B** A comparison of the mean function evaluations for each search strategy (the filled areas represent the standard error of these estimates). From here it is clear that the number of function evaluations required for random search grows exponentially with problem size, whereas for the genetic algorithm this number grows more slowly, approximately linearly.

| $ V $ | Random search      | Algorithm search   |
|-------|--------------------|--------------------|
| 3     | $5.39 \pm 0.62$    | $7.93 \pm 0.62$    |
| 4     | $15.52 \pm 1.77$   | $19.87 \pm 2.25$   |
| 5     | $60.55 \pm 8.53$   | $49.06 \pm 4.38$   |
| 6     | $174.82 \pm 22.34$ | $80.12 \pm 7.84$   |
| 7     | $407.15 \pm 54.45$ | $122.31 \pm 9.02$  |
| 8     | $723.96 \pm 83.43$ | $152.37 \pm 14.73$ |

Table 1: Mean numbers of function evaluations until encountering a satisfactory solution to the genetic inverter design problem. Estimates of the mean for random search and search using our algorithm are made for problem sizes (number of vertices available  $|V|$ ) from 3 to 8. The standard error of these estimates are also indicated. The random search is conducted as described in Methods.

rithm is performing the search efficiently, and suggest designing large genetic networks will be much more efficient with the genetic algorithm than with a random search.

Of particular interest is how the number of function evaluations required scales with the size of the problem. Figure 3B plots the data from Table 1 and shows clearly how random search grows much more quickly than the genetic algorithm as problem size increases. This is promising in the sense that the algorithm could be useful even for very large problem sizes, that is, for designing synthetic gene networks with a large number of interconnected genes, whereas random search quickly becomes intractable for larger networks.

### Genetic AND gate

The genetic inverter is a single input, single output logic gate and can be implemented with a very small genetic network. A slightly more complex genetic logic gate is the AND gate, which takes two inputs and produces a single high output when both inputs are high. The genetic AND has been implemented several times (Wang et al., 2011), but here of particular interest is the implementation using the gene network in Figure 4A, because this implementation only uses repressive transcription factors (Nielsen et al., 2016). The graph that encodes this gene network is shown in Figure 4B, so it can be represented as a solution in our genetic algorithm and will be used as a benchmark for a genetic AND gate.

This genetic AND gate is simulated and trajectories sampled for different combinations of the two inputs to produce the distributions in Figure 4C. The AND gate should highly express the output only when both inputs are high. In all other conditions output expression should be low. The distributions in Figure 4C conform to these rules, since the mean expression when both inputs are high (green dashed line) is higher than in all other cases. By constructing the difference distribution from the supposed high and supposed low output distributions, we can measure the SNR of this AND gate in the same way as for the genetic inverter. The result indicates disappointing performance with a high error rate, and

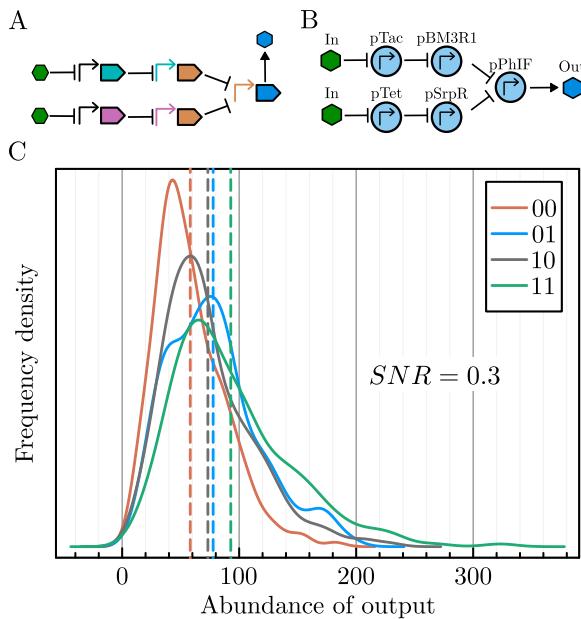


Figure 4: A human designed genetic network for a genetic AND gate. **A** A schematic diagram of the gene network implementing the AND gate, as it has previously been implemented in Nielsen et al. (2016). **B** The graph that encodes the AND gate, showing how the AND gate would be represented by the genetic algorithm as a solution. **C** Distributions of output gene expression sampled from stochastic simulations of the gene network in **A**. Each distribution corresponds to a particular combination of inputs. When one of the inputs is low (00, 01 and 10, orange, grey and blue), the output tends to be lower. When both outputs are high (11, green), the output tends to be higher. This is reflected in the means of these individual distributions (dashed lines) and in the proper function of the AND gate. The network is not perfect, the SNR of the difference distribution between high and low output states is only 0.3.

$SNR = 0.3$ . Perhaps the performance could be improved by tuning the parameters of the genetic components, or replacing them with different components. However, if this is not possible, it might be possible to redesign the topology of the network, using the existing genetic parts, and achieve better performance.

To generate such a design for a genetic AND with a higher SNR we asked the genetic algorithm to evolve a new solution using up to 7 promoters ( $|V| = 7$ ) and starting from a randomly generated solution. Figure 5 shows the results over 64 generations, over which the SNR of the proposed AND gates increased from  $-0.74$  to  $1.18$ , with the end result shown as a graph in the inset of Figure 5A that uses 6 out of 7 of the available promoters. This graph is more complex than that of the benchmark design and it is difficult to discern how it can operate as an AND gate. It seems unlikely a human would have ever generated such a network, but the results of stochastic simulations predict that it would perform very well as an AND gate. Figure 5B shows the results of stochastic simulations of the proposed solutions at several generations of the evolution process. The output expression distributions generated from sampling these simulation increasingly resemble an ideal AND gate, and at generation 64, the performance is around 4 times better than the benchmark. These results suggest that the genetic algorithm is capable of producing solutions that improve upon those designed by humans, at least in terms of the performance metric we have used here.

## Conclusion and Discussion

In this study we adapt CGP as a method for evolving designs for genetic networks which process their inputs into outputs in some specific and predetermined way. Using empirical measurements of the computational effort required to arrive at satisfactory solutions, we confirm that our algorithm significantly outperforms a random search of the solution space, especially as the size of that space grows.

For the problem of designing a genetic inverter, the algorithm is able to efficiently evolve solutions that are comparable in performance to a human generated benchmark design, as measured by stochastic simulation of the proposed solutions. In the second problem, designing a genetic AND gate, the algorithm can evolve designs which exceed the performance of the human designed benchmark. The workings of the proposed designs are difficult to interpret, but appears to perform around 4 times better than the benchmark.

Are logic gates and small sequential circuits the most complex computations that can be engineered in living systems? It looks like there are still unknown features of living matter that could be used to implement more complex synthetic systems, eventually realising what has been termed the cellular supremacy (Grozinger et al., 2019). Here, we focus on the topology of genetic networks as one of those features that deserves further attention.

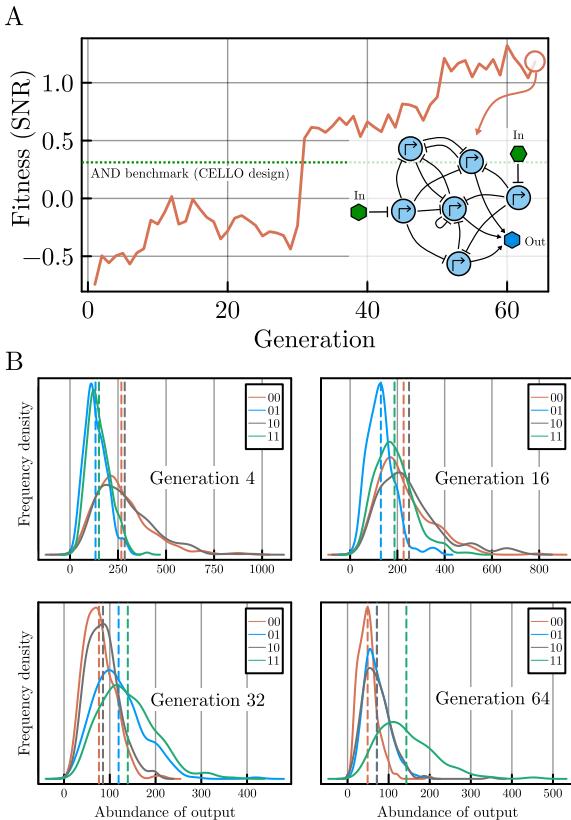


Figure 5: Evolution of a genetic AND gate using the genetic algorithm. **A** Fitness, measured in SNR of the difference distribution, of the evolved solution over 64 generations. Starting at  $SNR = -0.74$  at generation 1, fitness tends to plateau for some number of generations before increasing quickly in steps, at generations  $\approx 10$ ,  $\approx 30$  and  $\approx 50$ . At generation 64 the solution is the graph shown in the inset, and has  $SNR = 1.18$ . This solution is around 4 times better than our benchmark solution from Nielsen et al. (2016). **B** Snapshots of the distributions of output expression for the AND gates at various generations of the evolution process. At generations 4 and 16 the genetic network is clearly not performing AND of the inputs, although there is some improvement between the two. At generation 32 the AND logic has emerged and the distributions resemble those from Figure 4C. At generation 64 the performance of the solution has quite clearly improved upon the benchmark.

Whereas many existing genetic design frameworks focus on evolving parameters within fixed topologies, our proposed algorithm fixes the parameters and evolves new topologies, an approach which we see as complementary and which expands the space of possibilities for synthetic gene networks considerably. Genetic algorithms such as CGP are of special interest since they have been shown to generate interesting and non-intuitive solutions to engineering problems, for example antennas for communications subsystems on spacecraft (Hornby et al., 2006), and because they take inspiration from natural evolution. We expect that many complex gene network topologies could be evolved with these techniques that could not be conceived of by human engineers, but which might indeed already have been exploited by natural evolution. Comparison of our generated topologies with natural gene networks might yield interesting similarities.

Future work will deal with the experimental validation of the conclusions of this study by constructing the proposed gene networks. In terms of computational work, the given problems are examples chosen for this preliminary study, but the algorithm can in principle be used to design gene networks for many other purposes, including other genetic logic gates as well as networks with more varied dynamics such as bistable switches or oscillators. There will be opportunity to tune the hyperparameters  $MR$ ,  $\lambda$  and  $n$  in order to study and improve convergence properties.

Many extensions to the core idea presented here are also possible. For example, while initially we have considered the topology of synthetic networks within the volume of a single cell, we envision this work could be extended in the future to characterise networks of cells in distributed multicellular computations (Regot et al., 2011). In fact, as was true for single cell synthetic gene networks, synthetic consortia are often engineered following the same rules derived from electronic circuitry (Al-Radhawi et al., 2020; Goni-Moreno et al., 2011), while natural cellular populations have evolved complex social relationships and connections that can also be designed and modulated (Kong et al., 2018; Goni-Moreno and Amos, 2015).

## References

- Al-Radhawi, M. A., Tran, A. P., Ernst, E. A., Chen, T., Voigt, C. A., and Sontag, E. D. (2020). Distributed implementation of boolean functions by transcriptional synthetic circuits. *ACS Synthetic Biology*, 9(8):2172–2187.
- Amos, M. and Goñi-Moreno, A. (2018). Cellular computing and synthetic biology. *Computational matter*, pages 93–110.
- Andrianantoandro, E., Basu, S., Karig, D. K., and Weiss, R. (2006). Synthetic biology: new engineering rules for an emerging discipline. *Molecular systems biology*, 2(1):2006–0028.
- Appleton, E., Madsen, C., Roehner, N., and Densmore, D. (2017). Design automation in synthetic biology. *Cold Spring Harbor perspectives in biology*, 9(4):a023978.

- Beal, J., Goñi-Moreno, A., Myers, C., Hecht, A., de Vicente, M. d. C., Parco, M., Schmidt, M., Timmis, K., Baldwin, G., Friedrichs, S., et al. (2020). The long journey towards standards for engineering biosystems: Are the molecular biology and the biotech communities ready to standardise? *EMBO reports*, 21(5):e50521.
- Benenson, Y. (2012). Biomolecular computing systems: principles, progress and potential. *Nature Reviews Genetics*, 13(7):455–468.
- Calles, B., Goñi-Moreno, Á., and de Lorenzo, V. (2019). Digitalizing heterologous gene expression in gram-negative bacteria with a portable on/off module. *Molecular systems biology*, 15(12):e8777.
- Castle, S. D., Grierson, C. S., and Gorochowski, T. E. (2021). Towards an engineering theory of evolution. *Nature Communications*, 12(1):3326.
- de Las Heras, A., Fraile, S., and de Lorenzo, V. (2012). Increasing signal specificity of the tol network of *pseudomonas putida* mt-2 by rewiring the connectivity of the master regulator xylR.
- Del Vecchio, D. (2015). Modularity, context-dependence, and insulation in engineered biological circuits. *Trends in biotechnology*, 33(2):111–119.
- Goñi-Moreno, A. and Amos, M. (2012). Continuous computation in engineered gene circuits. *Biosystems*, 109(1):52–56.
- Goni-Moreno, A. and Amos, M. (2015). Discus: a simulation platform for conjugation computing. In *Unconventional Computation and Natural Computation: 14th International Conference, UCNC 2015, Auckland, New Zealand, August 30–September 3, 2015, Proceedings 14*, pages 181–191. Springer.
- Goñi-Moreno, Á., Benedetti, I., Kim, J., and de Lorenzo, V. (2017). Deconvolution of gene expression noise into spatial dynamics of transcription factor–promoter interplay. *ACS synthetic biology*, 6(7):1359–1369.
- Goni-Moreno, A., Redondo-Nieto, M., Arroyo, F., and Castellanos, J. (2011). Biocircuit design through engineering bacterial logic gates. *Natural Computing*, 10:119–127.
- Grozingher, L., Amos, M., Gorochowski, T. E., Carbonell, P., Oyarzún, D. A., Stoof, R., Fellermann, H., Zuliani, P., Tas, H., and Goñi-Moreno, A. (2019). Pathways to cellular supremacy in biocomputing. *Nature communications*, 10(1):5250.
- Hornby, G., Globus, A., Linden, D., and Lohn, J. (2006). Automated Antenna Design with Evolutionary Algorithms. In *Space 2006*. American Institute of Aeronautics and Astronautics. *eprint*: <https://arc.aiaa.org/doi/pdf/10.2514/6.2006-7242>.
- Kim, J., Goñi-Moreno, A., Calles, B., and de Lorenzo, V. (2019). Spatial organization of the gene expression hardware in *pseudomonas putida*. *Environmental microbiology*, 21(5):1645–1658.
- Kim, J., Goñi-Moreno, A., and de Lorenzo, V. (2021). Subcellular architecture of the xyl gene expression flow of the tol catabolic plasmid of *pseudomonas putida* mt-2. *Mbio*, 12(1):e03685–20.
- Kong, W., Meldgin, D. R., Collins, J. J., and Lu, T. (2018). Designing microbial consortia with defined social interactions. *Nature Chemical Biology*, 14(8):821–829.
- Miller, J. F. (1999). An empirical study of the efficiency of learning boolean functions using a Cartesian Genetic Programming approach. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2*, GECCO'99, pages 1135–1142, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Nielsen, A. A., Der, B. S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E. A., Ross, D., Densmore, D., and Voigt, C. A. (2016). Genetic circuit design automation. *Science*, 352(6281):aac7341.
- Nobile, M. S., Besozzi, D., Cazzaniga, P., Pescini, D., and Mauri, G. (2013). Reverse engineering of kinetic reaction networks by means of Cartesian Genetic Programming and Particle Swarm Optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 1594–1601. ISSN: 1941-0026.
- Regot, S., Macia, J., Conde, N., Furukawa, K., Kjellén, J., Peeters, T., Hohmann, S., De Nadal, E., Posas, F., and Solé, R. (2011). Distributed biological computation with multicellular engineered networks. *Nature*, 469(7329):207–211.
- Sanassy, D., Fellermann, H., Krasnogor, N., Konur, S., Mierla, L. M., Gheorghe, M., Ladroue, C., and Kalvala, S. (2014). Modelling and Stochastic Simulation of Synthetic Biological Boolean Gates. In *2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICES)*, pages 404–408.
- Tas, H., Grozinger, L., Goñi-Moreno, A., and de Lorenzo, V. (2021a). Automated design and implementation of a nor gate in *pseudomonas putida*. *Synthetic Biology*, 6(1):ysab024.
- Tas, H., Grozinger, L., Stoof, R., de Lorenzo, V., and Goñi-Moreno, Á. (2021b). Contextual dependencies expand the re-usability of genetic inverters. *Nature communications*, 12(1):355.
- Wang, B., Kitney, R. I., Joly, N., and Buck, M. (2011). Engineering modular and orthogonal genetic logic gates for robust digital-like synthetic biology. *Nature communications*, 2(1):508.