Networks: Emergence of Modularity

02-251 — Great Ideas in Computational Biology
Spring 2019

(Items marked (optional) are helpful but not required to understand.)

1. Emergent / Evolved / Desired Properties of Evolving Systems

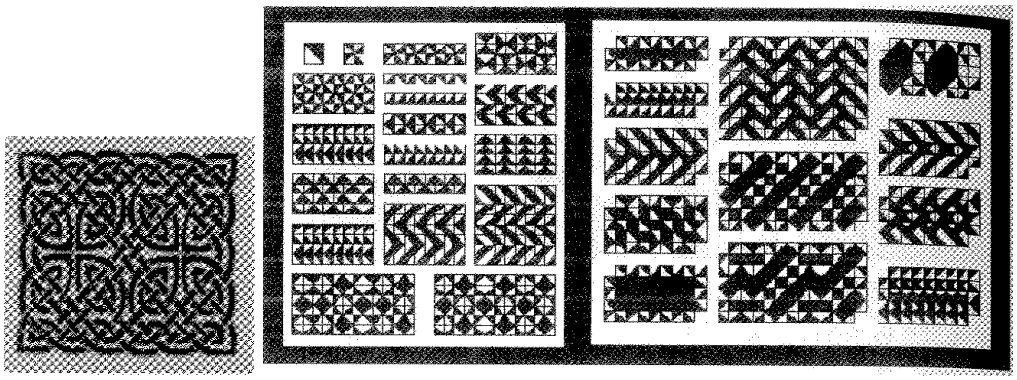
- Modularity: a network or system has high modularity if it can be cleanly separated into nearly independent components.
- Evolvability: system can adapt to changing environment.
- Robustness: system maintains function in the face of various types of failures (or attacks).

2. Modularity

- Purposes and effects of modularity:
  1. Creating reusable components:

     ```
     float euclidian(float *a, float *b, int n)
         { float sum = 0.0; for(int i=0; i < n; i++) sum += a[i] * b[i]; return sqrt(sum); }
     ```

  2. Isolating one operation or function from another [`sum` and `i` are local to the `euclidean` function.]
  3. Evolvability: can change quickly: `euclidean(a,b)` → `manhattan(a,b)`
  4. Compressibility: can encode modular systems more efficiently.
- Examples:
  1. Phenotypic modularity: "quasi independence" of phenotypic traits (Lewontin): changing your eye color doesn't affect the number of eyes you have — these traits are derived via semi-independent means. Important for natural selection to work well.
  2. Operons: units of bacterial transcription.
  3. Knots
  4. Patterns



Figures 12.3 and 12.7 from *Modularity: Understanding the Development and Evolution of Natural and Complex Systems* edited by Callebaut and Rasskin-Gutman.

  5. Developmental modularity: Mutant fruit flies created with eyes on their wings.

- Network "Modularity" (optional):

$$Q = \sum_{s=1}^{M} \left[\frac{m_s}{m} - \left(\frac{d_s}{2m} \right)^2 \right] \tag{1}$$

where M is the number of modules, m is the number of edges in the network, m_s is the number of edges in module s and d_s is the sum of the node degrees in module s. Q is the difference between observed number of edges within the groups and the number of intra-module edges expected based on the degrees alone.

3. Evolution of Modularity

- Many systems in biology and nature are intuitively modular. The basic MAPK pathway architecture, for example, is used repeatedly. Proteins are composed of domains that get reused, etc.
- Why did this modularity emerge? Was the modular solution the most efficient solution? The easiest to find? or did it have other benefits?
- One way to explore these questions: simulate evolution (including mutation and selection) in an idealized setting to see when/how/if modular structures emerge.
- Here we consider *digital circuits* composed of NAND gates. Digital circuits can be thought of as idealized models of regulatory and signaling pathways.
- A NAND ("not and") gate takes 2 Boolean inputs and outputs True if at least one input is False. NANDs are universal: you can construct any boolean function using only NAND gates. (Question: how would you construct a NOT gate? What about an OR gate?)
- Next few sections follow Kashtan and Alon. Spontaneous evolution of modularity and network motifs, PNAS 102(39):13773–13778, 2005.
- The first step will be to choose a procedure by which to simulate evolution.

4. Genetic Algorithms

- Genetic algorithms are used to solve optimization problems for which there are no good analytical solutions and where other more structured techniques (like integer programming) are not easy to apply.
- Here, Kashtan & Alon use a genetic algorithm as a scheme for simulating evolution.
- Biology → computer science → biology.
- In a genetic algorithm (GA), a population of *individuals* is tracked:

$$s_1, \ldots, s_p \tag{2}$$

- Each individual encodes a (probably non-optimal) solution to the problem using some representation. Designing this representation can effect the speed and accuracy of the GA.
- The basic GA algorithm. At each time step:
  1. A fitness $f(s_i)$ is computed for each individual. $f(s_i)$ would usually be a measure of how good a solution individual s_i encodes. The designer of the GA decides on f.

2. The more fit individuals are preferentially randomly selected to persist into the next generation. In other words, individuals with higher f values are more likely to be selected. This simulates natural selection.

3. Some of the individuals in the next generation are mutated or combined and the population is updated. The GA designer typically chooses some ways to mutate/combine the individuals.

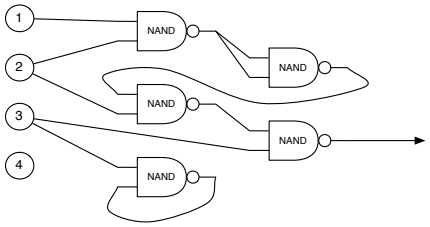
There are a *huge* number of variations of this basic idea. There are particularly a lot of variants of Step 2, e.g.: (a) select and duplicate the top K most fit individuals; (b) randomly sample from population with replacement using normalized fitnesses as a probability distribution; (c) select k individuals uniformly at random, and duplicate the most fit among them.

- The algorithm designer selects some *mutation* operations and their probabilities of occurrence.
- Some so called "cross-over operations" may also be applied where ≥ 2 individuals are combined. This simulates sexual reproduction.

5. Evolving Modular Circuits

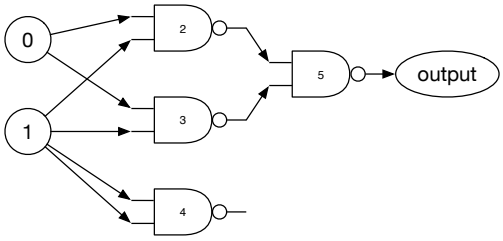
Here's an example of this idea applied to evolving circuits, the application explored in Kashtan & Alon.

- Population: 1000 circuits with < 12 NAND gates each, plus 4 input nodes and one output gate. Very few restrictions on the circuit: Can have self-loops, feedback looks, ignored inputs, gate outputs sent to any number of other gates, etc. E.g.:



Much is left unspecified about how circuits would be evaluated in the presence of loops.

- Encoding: Each s_i is encoded as a "genome". Encoding for circuit design could be the a sequence of codes that indicate the inputs to each gate, with 1 "gene" for each gate, and 1 gene for the output. E.g. if we limited ourselves to 4 gates, the encoding of the left circuit could be: 010111235, as shown at right:



Gate1	Gate2	Gate3	Gate4	Output
0,1	0,1	1,1	2,3	5

(This is similar to the representation used by Kashtan & Alon.)

- Fitness: Fraction of the 2^4 possible inputs for which the circuit computes the following function:

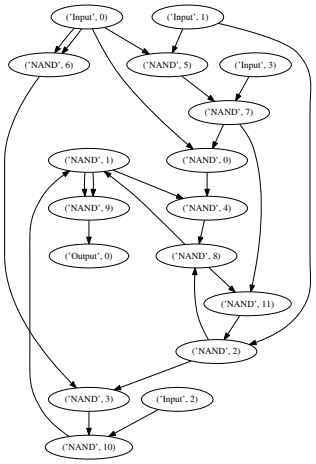
$$(a \textbf{ xor } b) \textbf{ and } (c \textbf{ xor } d) \tag{3}$$

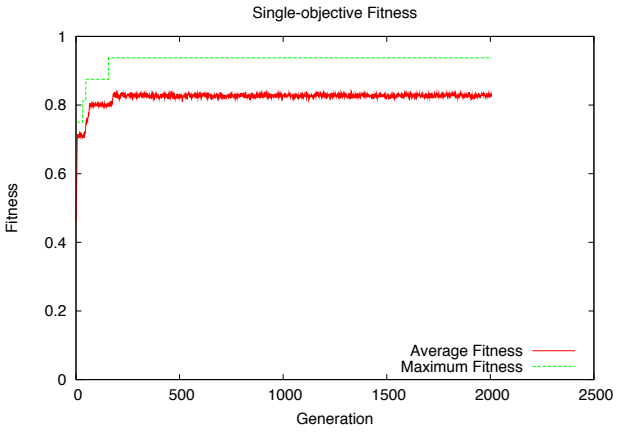
- Mutation: With probability 0.7 a single edge in a circuit is rewired (always keeping # of inputs for every gate = 2)

- Selection: At each step, the 300 most fit individuals are copied without change. The bottom 300 individuals are replaced with potentially mutated copies of the top 300 individuals. The middle 400 individuals are copied and potentially mutated.

6. GA Finds Good — but non-modular — solutions when given a single goal

- Circuit computing exactly 3 found within $\approx 10,000$ generations most of the time.
- Very low normalized modularity $Q \approx 0.12$.
- Even if the starting circuit population was chosen to be identical highly modular solutions, the modularity rapidly decreases.
- Optimal solution with a single objective function. Here, we're using my implementation not Kashtan & Alon's, so the behavior is slightly different. Does not have perfect fitness after ≈ 2000 generations





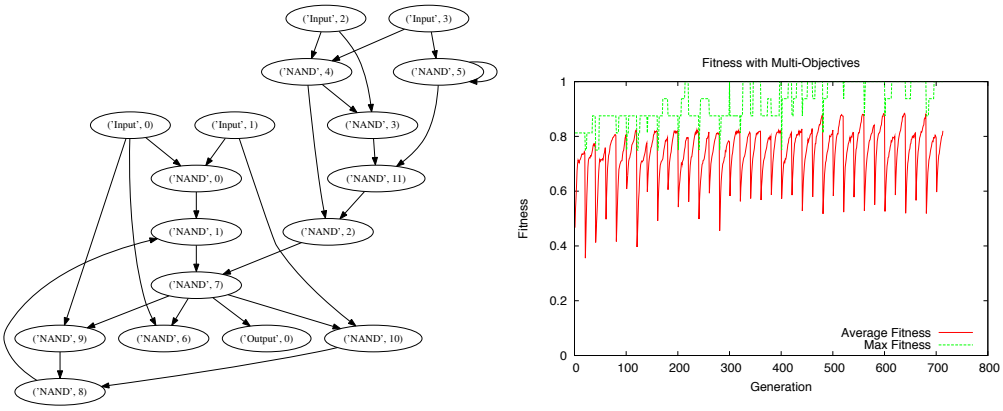
7. When the goal is varied, modularity emerges

- Plausible to assume real world environment has changing goals.
- Every 20 generations, the fitness function (goal) was alternated between (3) and:

$$(a \textbf{ xor } b) \textbf{ or } (c \textbf{ xor } d) \tag{4}$$

  The difference between (3) and (4) is the and/or operator in the middle.
- When the "environment" (aka goal, aka fitness) frequently changes, Kashtan & Alon observe:
  1. Normalized modularity is much higher: $Q \approx 0.54$.
  2. Evolved circuits can readily switch between the two goals, adapting after ≈ 5 generations. This is because the evolved solution requires only 2 edges to switch between the selected goals.
  3. Circuits that compute either goal perfectly are observed much sooner.

     Results from reproducing the Kashtan & Alon experiment (with slight parameter differences):



Left is a most fit circuit, right is the trace of how fitness changes over time.

4. If the two goals were *non-modular* (e.g. two random boolean functions), the resulting circuits were not modular.

- "Meta" fitness effect: more modular → more adaptable to changing environment → more fit you will be.
- Because varied goals were modular, high modularity → high evolvability.

8. To the biology!

- Kreimer et al. The evolution of modularity in bacterial metabolic networks. PNAS 105(19):6976–6981, 2008.
  1. Metabolic networks from KEGG: Nodes = enzymes. Edges between enzymes that catalyze the same reaction, and between all pairs of enzymes in different reactions R_1 and R_2 if a product of R_1 is a substrate of R_2.
  2. Modularity assessed via Newman's algorithm, which is a (good) heuristic but makes no guarantees.
  3. A few examples of obligate pathogens with complex life cycles (Rickettsia and Borrelia) show higher than expected modularity. Assumption is that being an obligate pathogen means your environment will change often.
  4. Similarly, host-associated organisms have small modularity scores in general, with the supposed explanation that they are finely tuned to their (unchanging) host environment.
  5. Not everything has an explanation: endosymbionts are more modular than pathogens, e.g.
  6. Ancestral networks (inferred using a simplistic parsimony criteria) are predicted to be more modular, which they hypothesize may be due to increased specialization over time.
- Parter M, Kashtan N, Alon U (2007) Environmental variability and modularity of bacterial metabolic networks. BMC Evol Biol 7:169195.
  1. 117 bacterial metabolic networks (KEGG, node = metabolite, edge = reaction, highly connected metabolites removed)
  2. Divided into classes based on the type / environment of the species: obligate (require association with a host), specialized (live in highly "specialized" environments), aquatic, facultative (bacteria often, but not necessarily, associated with a host), multiple, and terrestrial (soil).
  3. Variability: obligate < specialized < aquatic < facultative < multiple < terrestrial. # Transcription factors also used as a proxy for environmental variability (since more regulation indicates the

need more fine-tuned responses to the environment) and this agrees with the ordering of these classes.

4. Modularity increases with these qualitative classes.

9. Varying goals $\Longrightarrow$ faster evolution

- Even though a optimization problem with varying goals seems harder (design circuits that can handle 2 different expressions), it is more easily solved by the GA. The likely explanation here is that the varying goals force the population out of local optima.
- Explored in more detail in: Kashtan N, Noor E, Alon U (2007) Varying environments can speed up evolution. Proc Natl Acad Sci USA 104:1371113716.
- Scenarios:
  1. FG: fixed goal
  2. MVG: cycle through various modular goals.
  3. RVGC: alternate between modular goal & a single random goal.
  4. RVGV: alternate between modular goal & constantly changing random goals.
  5. VG0: alternate between modular goal & neutral evolution (no goal)
- The more generations it takes to optimize a function, the bigger the speedup of MVG vs. FG
- RVGV also lead to faster optimization; other goals $\rightarrow$ slower optimization.
- Speed-ups observed for a wide range of switching intervals.

10. Another model — similar results (optional)

- An alternative model: organisms transform environmental resources into required products.
- Lipson et al. On the Origin of Modular Variation. *Evolution*, 56(8), 2002, pp. 15491556.
  1. Individual: Matrix A (elements $-1, 0, 1$)
  2. Environment: ± 1 Vector E.
  3. Goal: ± 1 Vector F.
  4. Fitness: $|F - AE|$.
  5. Proxy for "modularity": Coupling = fraction of nonzero entries of A. Inversely correlated with modularity.
- Details: A is 8×8; F, E set randomly to ± 1; mutations: change an element of A at random; repeated select a fit individual; mutate it; replace an individual selected inversely proportional to its fit. Randomly flipped a sign of E after every θ generations.
- Variation $\rightarrow$ lower fitness, but higher modularity (aka lower coupling).
- Lower modularity $\rightarrow$ longer time to respond to environmental changes.
- Note major difference from Kashtan & Alon: Lipson et al's goals do not vary modularly.