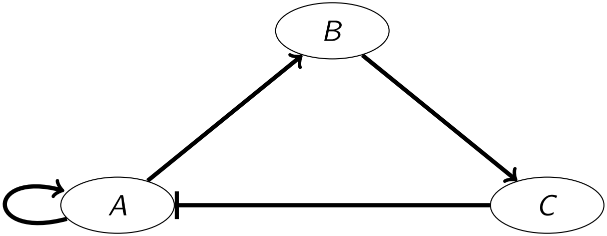
Parameter Space Decomposition of Regulatory Networks with Multiple Thresholds

Adam Zheleznyak
Mentors: Marcio Gameiro and Konstantin Mischaikow

CoSP Student Workshop
July 29, 2020

Regulatory Networks

Collection of species (DNA, RNA, proteins) interacting:



We want to understand the kinds of dynamics we can get, but this is difficult as there are many different parameters that can vary:

- How much one species affects another
- When each species starts to affect another
- The decay rate of each species

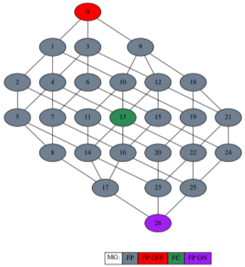
DSGRN[1]

My mentors have developed a computational technique that makes analyzing the dynamics of regulatory networks feasible and have created the software *Dynamic Signatures Generated by Regulatory Networks* (DSGRN).

In: Regulatory Network

Out: Parameter Graph

$$\begin{tikzcd}
 & B \arrow[dr] & \\
A \arrow[ur] \arrow[loop left] & & C \arrow[ll, dash, "\dashv"']
\end{tikzcd}
\longrightarrow$$



[1] Bree Cummins, Tomas Gedeon, Shaun Harker, Konstantin Mischaikow, and Kafung Mok. *Combinatorial Representation of Parameter Space for Switching Systems*. SIAM Journal on Applied Dynamical Systems, 15 (2016).

Mathematical Definition of a Regulatory Network

In DSGRN, this is how a regulatory network is modeled:

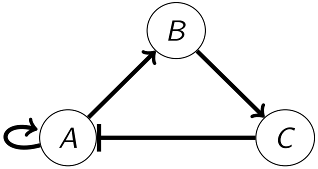
Variables: $x_1, \ldots, x_N \in \mathbb{R}_{>0}$

Parameters:

- Each species has a decay rate γ_j
- Each edge has a low value $\ell_{j,i}$, high value $\ell_{j,i} + \delta_{j,i}$, and threshold $\theta_{j,i}$
- Each $\gamma_j, \ell_{j,i}, \delta_{j,i}, \theta_{j,i} \in \mathbb{R}_{>0}$

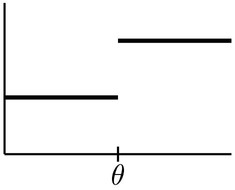
Differential equations: $\dot{x}_j = -\gamma_j x_j + \Lambda_j(x)$, where

$$\Lambda_j(x) = \sum_{i \to j} \begin{cases} \ell_{j,i} & x_i < \theta_{j,i} \\ \ell_{j,i} + \delta_{j,i} & x_i > \theta_{j,i} \end{cases} + \sum_{i \dashv j} \begin{cases} \ell_{j,i} + \delta_{j,i} & x_i < \theta_{j,i} \\ \ell_{j,i} & x_i > \theta_{j,i} \end{cases}$$

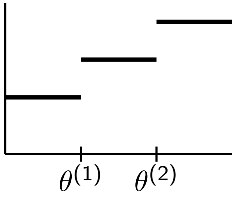


My Work

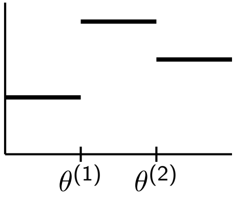
For my project, I worked on generalizing the interactions between species, allowing there to be multiple thresholds:
For some *m*, each edge has $\theta^{(1)}, \dots, \theta^{(m-1)}$ as thresholds and $\ell, \delta^{(1)}, \dots, \delta^{(m-1)}$ as the expression levels. So instead of:



we might have:



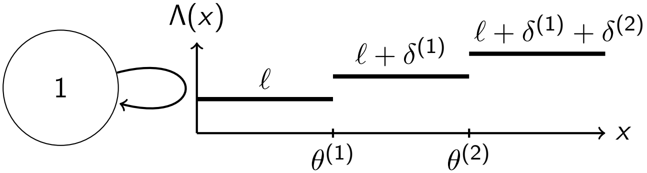
or



Parameter Space Decomposition

Each parameter in DSGRN is defined by a set of inequalities. This allows for a very quick computation to find the dynamics of a given parameter.

For example, let's take a simple regulatory network:

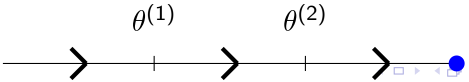


Recall: $\dot{x} = -\gamma x + \Lambda(x)$

Then one possible parameter is

$$\gamma\theta^{(1)} < \gamma\theta^{(2)} < \ell < \ell + \delta^{(1)} < \ell + \delta^{(1)} + \delta^{(2)}$$

which gives this phase line:



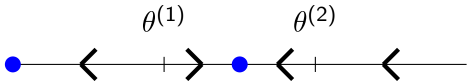
Parameter Space Decomposition

Recall: $\dot{x} = -\gamma x + \Lambda(x)$

Another parameter is

$$\ell < \gamma\theta^{(1)} < \ell + \delta^{(1)} < \ell + \delta^{(1)} + \delta^{(2)} < \gamma\theta^{(2)}$$

which gives a different phase line:



Not all parameters are possible. For example we can't have

$$\ell < \gamma\theta^{(1)} < \ell + \delta^{(1)} + \delta^{(2)} < \gamma\theta^{(2)} < \ell + \delta^{(1)}$$

because $\ell + \delta^{(1)} + \delta^{(2)} < \ell + \delta^{(1)}$ implies $\delta^{(2)} < 0$ which is not allowed. Thus we must find the possible parameters and this becomes computationally difficult with more complicated systems.

Parameter Space Decomposition (PSD)

If we want DSGRN to be able to handle regulatory networks with multiple thresholds, we need to compute the parameter space decomposition i.e. find all possible parameters as defined by inequalities. To put it formally:

Let n be the number of input edges at a given node, and let each edge have m levels of expression (i.e. $m-1$ thresholds).
Let $E := \{\alpha : \{1, 2, \ldots, n\} \to \{0, 1, \ldots, m-1\}\}$ be the set of *m-ary functions on* $\{1, \ldots, n\}$. Then we define

$$\mathcal{P} := \left\{ \sum_{i=1}^{n} (\ell_i + \sum_{k=1}^{\alpha(i)} \delta_i^{(k)}) : \alpha \in E \right\}$$

$$\mathcal{P} \subset \mathbb{R}[\ell_1, \ldots, \ell_n, \delta_1^{(1)}, \ldots, \delta_n^{(1)}, \ldots, \delta_1^{(m)}, \ldots, \delta_n^{(m-1)}]$$

which is all the possible values $\Lambda(x)$ can hold.

The PSD problem is to find all possible orderings of elements in $\mathcal{P}$ that are *admissible*.

Parameter Space Decomposition (PSD)

An *admissible* ordering means that there is some set of parameters that satisfies the ordering. Formally:

Say the polynomials in $\mathcal{P}$ are indexed in some way from 1 to $|\mathcal{P}|$.
For σ in the symmetric group $S_{|\mathcal{P}|}$, we call σ *admissible* if its realizable set

$$\Xi_\sigma := \{\xi \in \Xi : p_{\sigma(k)}(\xi) < p_{\sigma(k+1)}(\xi) \text{ for all } 1 \leq k \leq |\mathcal{P}|\}$$

is non-empty.

In other words, the order $\prec_\sigma$ can be realized where

$$p_{\sigma(1)} \prec_\sigma p_{\sigma(2)} \prec_\sigma \cdots \prec_\sigma p_{\sigma(|\mathcal{P}|)}.$$

LC-LEP[2]

Because the polynomials in $\mathcal{P}$ are linear, the PSD problem can be stated as an instance of a more general problem, the *linearly constrained linear extension problem* (LC-LEP):

Let $\mathcal{P} = \{p_0, \dots, p_K\}$ be a set of linear polynomials defined on a linearly constrained region $\Xi \subset \mathbb{R}^d$ (Ξ is called the evaluation domain).

Then let $\prec$ be the partial order on $\mathcal{P}$ such that if $p \prec q$ then

$$p(\xi) < q(\xi) \text{ for all } \xi \in \Xi.$$

Then the solution to LC-LEP is to find all admissible total orders $\sigma \in S_{K+1}$. Because $\prec_\sigma$ must satisfy $\prec$, $\prec_\sigma$ is a *linear extension* of $\prec$.

The solution to LC-LEP is denoted $\mathcal{T}(\mathcal{P}, \prec, \Xi)$.

[2]Shane Kepley, Konstantin Mischaikow, Lun Zhang. *Computing linear extensions for Boolean lattices with algebraic constraints*.

Representation Vectors

Now I will go over an algorithm that can solve LC-LEP in a reasonable amount of time, and this algorithm can then be used for the PSD problem.

But first, there are some constructions we must define:

Given a linear polynomial p, there exists the *representation vector* u_p where

$$p(\xi) = \xi \cdot u_p \text{ for all } \xi \in \mathbb{R}^d.$$

Additionally, recall that Ξ is linearly constrained which means there is a set of linear polynomials $\mathcal{Q}_\Xi$ such that

$$\Xi = \{\xi \in \mathbb{R}^d : \xi \cdot u_q > 0 \text{ for all } q \in \mathcal{Q}_\Xi\}.$$

Cones

The region $C \subset \mathbb{R}^d$ is called a *cone* if for any $v \in C$ and $\theta \in [0, \infty)$, $\theta v \in C$. We call a cone C *pointed* if it is closed, convex, and satisfies

$$C \cap -C = C \cap \{-v : v \in C\} = \{0\}.$$

A vector v is called a *conic combination* of vectors $v_1, \ldots, v_k$ if there exists $\theta_1, \ldots, \theta_k \geq 0$ such that $v = \theta_1 v_1 + \cdots + \theta_k v_k$.

The set of all conic combinations of a set $V = \{v_1, \ldots, v_k\}$ is called the *conic hull* and is given by

$$\text{cone}(V) := \{\theta_1 v_1 + \cdots + \theta_k v_k : \theta_i \geq 0, i = 1, \ldots, k\}$$

cone(V) is always a closed and convex set.

Fact about cones

Proposition

Let $V = \{v_1, \ldots, v_m\} \subset \mathbb{R}^d$ and suppose cone(V) is pointed. If $-v \notin$ cone(V), then cone$(V \cup \{v\})$ is pointed.

Proof: Suppose cone$(V \cup \{v\})$ is not pointed. Then there exist $w \neq 0$ such that $w, -w \in$ cone$(V \cup \{v\})$ i.e.

$$w = \sum_{i=1}^{m} \alpha_i v_i + \alpha v \text{ and } -w = \sum_{i=1}^{m} \beta_i v_i + \beta v$$

where all $\alpha_i, \alpha, \beta_i, \beta \geq 0$. If $\alpha = \beta = 0$, then $w, -w \in$ cone(V) which contradicts the fact cone(V) is pointed. If we add the two equations above and rearrange, we get

$$-(\alpha + \beta)v = \sum_{i=1}^{m} (\alpha_i + \beta_i) v_i$$

which contradicts the fact that cone(V) is pointed.

Preliminary Algorithms

To check if a vector is in a cone, this can be stated as a linear feasibility problem:

$$\begin{aligned} \text{Does there exist} \quad & \alpha \\ \text{such that} \quad & \mathbf{V}\alpha = v \\ \text{and} \quad & \alpha \geq 0? \end{aligned}$$

Linear feasibility is a well studied problem with many algorithms available, so we can just pick one and have an algorithm for cone inclusion, which we will just call `InCone`.

Preliminary Algorithms

Then, using the previous result, we can write an algorithm to check if a cone is pointed

```
Function CheckCone(V):
    $V' = \{v_1\}$
    for $i = 2 \dots m$ do
        if InCone($-v_i, V'$) then
            Return False
        else
            $V' = V' \cup \{v_i\}$
        end
    end
    Return True
End Function
```

Algorithm for LC-LEP

Let $V_{\Xi} := \{u_q : q \in \mathcal{Q}_{\Xi}\}$ and

$$V_{\prec} := \{u_{p-q} : u_{p-q} \text{ represents } p - q \text{ where } q \prec p, q, p \in \mathcal{P}\ \}.$$

Define the *base cone* $V_0 := V_{\Xi} \cup V_{\prec}$

Because $u_q \cdot \xi > 0, \forall \xi \in \Xi$ for each $q \in \mathcal{Q}_{\Xi}$ (by definition of $\mathcal{Q}_{\Xi}$) and $u_{p-q} \cdot \xi = p(\xi) - q(\xi) > 0, \forall \xi \in \Xi$ (by definition of $\prec$), if we use `CheckCone` must say V_0 is pointed since at each step of the algorithm, if $-v \in \text{cone(V)}$ for some $v \in V_0$, then $\xi \cdot (-v) < 0$. However this gives a contradiction since if $-v = \sum_{i=1}^{m} \alpha_i v_i$ for $v_i \in V_0$, then

$$\xi \cdot (-v) = \sum_{i=1}^{m} \alpha_i \xi \cdot v_i > 0.$$

Algorithm for LC-LEP

Proposition

Suppose $V = \{v_0, \ldots, v_m\} \subset \mathbb{R}^d$ is a collection of nonzero vectors such that cone(V) is a pointed cone. Then, there exists some $v' \in \mathbb{R}^d$ such that $v' \cdot v_i > 0$ for all $0 \leq i \leq m$.

Given $\sigma \in S_{K+1}$, define
$V_\sigma := V_0 \cup \{u_{p_{\sigma i+1}} - u_{p_{\sigma i}} : p_{\sigma i} \in \mathcal{P}, i = 0, \ldots, K-1\}$

Proposition

For $\sigma \in S_{K+1}$, $\Xi_\sigma \neq 0$ if and only if cone(V_σ) is pointed.

Proof: Assume $\Xi_\sigma \neq \emptyset$ and that $\xi \in \Xi_\sigma$. If cone(V_σ) is not pointed, then there are nonzero vectors $-v, v \in$ cone(V_σ). But by definition of Ξ_σ means $-v \cdot \xi > 0$ and $v \cdot \xi > 0$ which is a contradiction.

If cone(V_σ) is pointed, then by the first proposition, there is some ξ such that $\xi \cdot v > 0$ for all $v \in V_\sigma$ which by definition implies $\xi \in \Xi_\sigma$ i.e. $\Xi_\sigma \neq 0$.

Algorithm for LC-LEP

By the last proposition, we have the equivalence

$$\mathcal{T}(\mathcal{P}, \prec, \Xi) = \{\sigma : \Xi_\sigma \neq \emptyset\} = \{\sigma : \text{cone}(V_\sigma) \text{ is pointed}\}$$

Algorithm for LC-LEP

```
Input: $\sigma_{\text{part}} = [\,], \mathcal{P}, V = V_0, \texttt{Ret} = \{\}$
Output: $\mathcal{T}(\mathcal{P}, \prec, \Xi)$
Result: Ret: collection of all linearly realizable total order under restriction of $V$
1  Function OrderingGenerator($\sigma_{\text{part}}, \mathcal{P}, V, \texttt{Ret}$):
2      if $\sigma_{\text{part}} == [\,]$ and CheckCone($V$) is not True then
3          Return
4      end
5      $l + 1 =$ length of $\sigma_{\text{part}}$
6      if $l == K$ then
7          add $\sigma_{\text{part}}$ to Ret
8          Return
9      end
10     for $i = 0 \,..\, K$ do
11         if $i \notin \sigma_{\text{part}}$ then
12             $\mathbf{u}' = \mathbf{u}_{p_i} - \mathbf{u}_{p_{\sigma_{\text{part}}(l)}}$
13             if not InCone($-\mathbf{u}', V$) then
14                 OrderingGenerator($\sigma_{\text{part}} + [i], \mathcal{P}, V \cup \{v'\}, \texttt{Ret}$)
15             end
16         end
17     end
18 End Function
```

Results So Far

With the algorithm, I calculated a database of total orders.

Here, n is the number of inputs and $m-1$ is the number of thresholds.

$(n = 2,\ m = 2)$: 2 total orders (instant)
$(n = 3,\ m = 2)$: 12 total orders (instant)
$(n = 4,\ m = 2)$: 336 total orders ($\sim$ 90 seconds)
$(n = 2,\ m = 3)$: 36 total orders (instant)
$(n = 2,\ m = 4)$: 6660 total orders ($\sim$ 5 minutes)

Each of $(n = 3,\ m = 3)$ and $(n = 2,\ m = 5)$ cases didn't finish after 8 hours on my laptop.

Next Steps

There is still more work to be done:

- Calculate the total orders for larger n and m on a server using distributed CPUs.
- Modify the code of DSGRN so that it can handle multiple thresholds in order to use the database I've created.

Acknowledgements

This work was carried out while I was a participant in the 2020 DIMACS REU program at Rutgers University, supported by NSF HDR TRIPODS award CCF-1934924.

Thank you to Lazaros Gallos and Parker Hund for organizing the REU.

I'd also like to acknowledge the support and guidance of Marcio Gameiro, Shane Kepley, and Konstantin Mischaikow. Thank you for making the summer great!