Constrained Coding
The State-Splitting Algorithm

Han-Mo Ou
Evangelos Tsiamalos

Constrained System
State-Splitting
Finite-State Encoder
Approximate Eigenvectors
State-splitting Algorithm

Background: Labeled Graph and Constrained System

- A constrained system $\mathcal{S}$ can be described by a labeled graph $H(V, E, L)$
  - vertices $v \in V$ are called "states"
  - **edges** $e \in E$ are labeled with the code
  - each edge is labeled with the symbol
- Note: the labeled graph is not unique

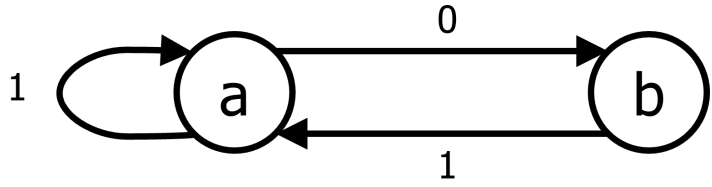


Figure: Labeled Graph - cannot have consecutive 0s

Background: Graph Powers

- the q^{th} power of a graph $H = (V, E, L)$, H^q, is the graph with same set of states V, but one edge of each path of length q in H labeled by the q-block generated by that path

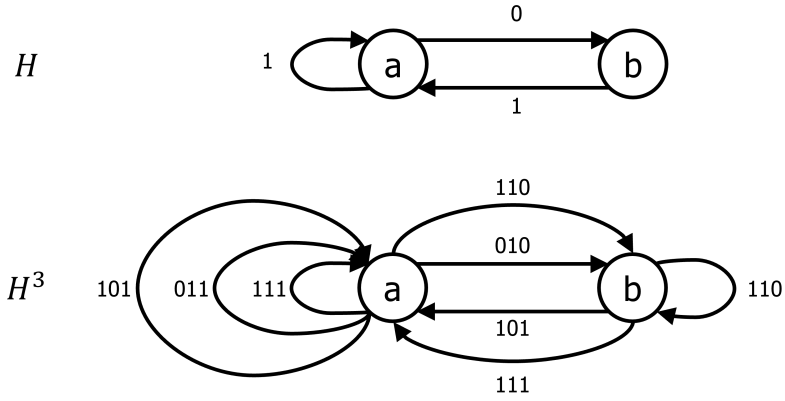


Figure: Labeled graph and its cube

Background: Decodability

- Anticipation of a graph: the smallest N such that any two paths of length $N+1$ with same initial state and labeling must have same initial edge
  - to decode, we want the graph to have **finite anticipation**
- Definite: a graph is (m, a)-definite if given any word $w_{-m} \dots w_0 \dots w_a$, the set of paths $e_{-m} \dots e_0 \dots e_a$ that generate the word would agree on e_0
  - a stronger condition than finite-anticipation

Background: Capacity of System

- Capacity of system: defined as $cap(\mathcal{S}) = \limsup_{\ell \to \infty} \frac{1}{\ell} \log_2 N(\ell, \mathcal{S})$
  - $N(\ell, \mathcal{S})$ is number of words of length ℓ in $\mathcal{S}$
- Finite state coding theorem (Theorem 4.1): for integers p, q, if $\frac{p}{q} \leq cap(\mathcal{S})$, there exists a rate $p : q$ finite state encoder for $\mathcal{S}$ with finite anticipation
- How do we find such an encoder? $\rightarrow$ State-splitting

State (out-)Splitting

- For a labeled graph $H = (V, E, L)$, splitting at state u is determined by partitioning the set E_u of outgoing edges of u into $E_u = E_u^{(1)} \cup E_u^{(2)}$
- Resulting graph $H' = (V', E', L')$
  - u partitioned into $u^{(1)}$ and $u^{(2)}$
  - The new set of vertices would be $V' = (V - \{u\}) \cup \{u^{(1)}, u^{(2)}\}$
  - Incoming edges to u would produce two edges to $u^{(1)}$ and $u^{(2)}$
  - Outgoing edges from u would belong to one of the descendant states according to the partition

State Splitting Example

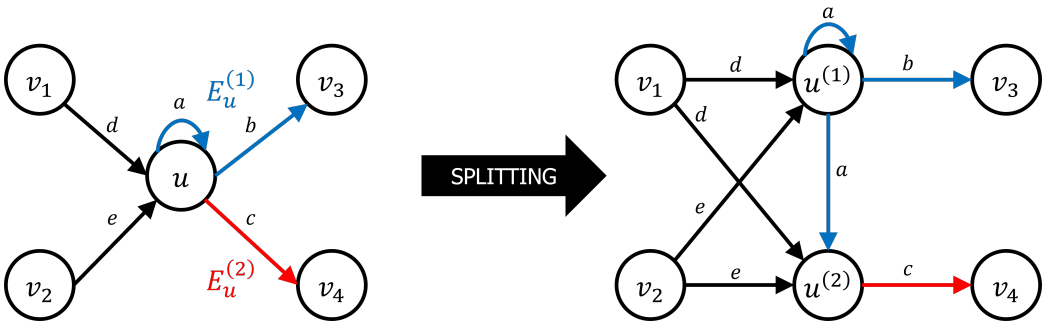


Figure: State-splitting example

State Splitting Properties

- Both H and H' represent the same constrained system
- The anticipation of H' is at most 1 above that of H
  - Finite iterations of state splitting preserve the finite anticipation property of a graph
- If H is (m, a)-definite, H' is $(m, a+1)$-definite

Goal

We start with a constraint system that is described by a deterministic graph $\boldsymbol{G}$.

We want to construct a finite state encoder.

Encoder

Take some constraint system S. We consider an (S, n) encoder which is an encoder that can be represented as a graph where the outgoing edges are assigned distinct input labels (or tags) from an alphabet of size n and the output labels (not necessarily distinct) correspond to some word in S.

From state u to state v, $u \xrightarrow{s/a} v$ means that we have input label s and output label a.

A rate $p : q$ finite state encoder for S is a tagged $(S^q, 2^p)$ encoder.

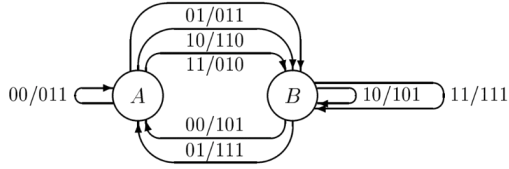
This means that we are going to use blocks of size q to encode a word of length p of the message.

Example

The $(0, 1)$-runlength-limited constraint.

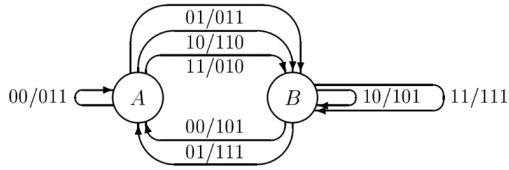
- the runs of 0s have length at most one
- we can have arbitrarily long runs of 1s

An example of a $2 : 3$ finite state encoder is:



Source: Marcus, Brian H., Ron M. Roth, and Paul H. Siegel. "An introduction to coding for constrained systems." Lecture notes (2001)

Example

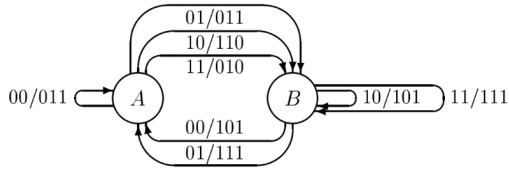


Source: Marcus, Brian H., Ron M. Roth, and Paul H. Siegel. "An introduction to coding for constrained systems." Lecture notes (2001)

We can use this graph as follows:

- Start from any one of the states, either (A) or (B). Let's say that we start at (A).
- If we follow any of the outgoing edges, we get a block of length 3 that satisfies the constraints of the system.
- We can encode the message by using the input labels which are binary strings of length 2.

Example



Source: Marcus, Brian H., Ron M. Roth, and Paul H. Siegel. "An introduction to coding for constrained systems." Lecture notes (2001)

For example, if we have the message

$$011000$$

that we want to encode using (0,1)-RLL we can use the graph to encode it as follows :

- Start at state (A)
- We encode "01" as "011" using the edge $A \xrightarrow{01/011} B$
- We encode "10" as "101" using the edge $B \xrightarrow{10/101} B$
- We encode "00" as "101" using the edge $B \xrightarrow{00/101} A$

So the encoded message is "011101101" which satisfies the (0,1)-RLL constraint.

Build an Encoder

- Start from the graph $\boldsymbol{G}$ of the constraint system.
- Take the $q-$power $\boldsymbol{G}^q$
- Iteratively modify it (state splitting)
- Until we end up with a subgraph with minimum degree of out-edges at least 2^p

Approximate Eigenvectors

Definition

We have a non-negative integer matrix $\boldsymbol{A}$ and integer n, an $(\boldsymbol{A}, n)-$approximate eigenvector $\boldsymbol{x}$ is a non-negative integer vector:

$$\mathbf{A}\mathbf{x} \geq n\mathbf{x}$$

In the Scope of a Graph

- $\boldsymbol{A}$ is the adjacency matrix
- Think of $\boldsymbol{x}$ as a vector of state weights

$$\boldsymbol{x} = (x_u)_{u \in V_G}$$

- The vector $\boldsymbol{x}$ is an $(A, n)-$ eigenvector if

$$\sum_{e \in E_u} x_{\tau_G(e)} \geq n x_u$$

for every state u.

Important Property

For a Graph $\boldsymbol{G}$,

- the all-one vector $\mathbf{1}$ is an $(\boldsymbol{A}_G, n)-$approximate eigenvector $\Leftrightarrow$ the graph has minimum out-degree at least n
- a $0-1$ vector is an $(\boldsymbol{A}_G, n)$-approximate eigenvector $\Leftrightarrow$ there is a sub-graph with minimum out-degree at least n

For a given Graph, are there approximate eigenvectors?

There exists an (A, n) approximate eigenvector *if and only if* $\lambda(A) \geq n$

Computing Approximate Eigenvectors

The Franaszek Algorithm

1: **Input**: non-negative integer matrix A, positive integer n
2: $y \leftarrow \xi$ # Initialization
3: $x \leftarrow 0$
4: **while** $x \neq y$ **do**
5: $x \leftarrow y$
6: $y \leftarrow \min\{\lfloor \frac{1}{n} Ax \rfloor, x\}$
7: **end while**
8: **return** x

where $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\xi}$ are vectors, $\lfloor \cdot \rfloor$ denotes the floor function and it acts component-wise on a vector $\lfloor \boldsymbol{x} \rfloor = \begin{pmatrix} \lfloor x_1 \rfloor \\ \vdots \\ \lfloor x_k \rfloor \end{pmatrix}$

Some Notes on the Algorithm

We observe the following:

- If we have two vectors $\boldsymbol{x}, \boldsymbol{y}$ that are $(\boldsymbol{A}, n)$ approximate eigenvectors, then $\boldsymbol{z} = (z_u)_{u \in V}$ with $z_u = max\{x_u, v_u\}$ is also an approximate eigenvector.
- So if we fix some vector $\boldsymbol{\xi}$
  - There may be no approximate eigenvector $\boldsymbol{x}$ with $\boldsymbol{x} \leq \boldsymbol{\xi}$
  - There exists some approximate eigenvector that is the "largest" approximate eigenvector $\boldsymbol{x}$ with $\boldsymbol{x} \leq \boldsymbol{\xi}$

Some Notes on the Algorithm

The initialization of the algorithm $\boldsymbol{\xi}$ matters.

If we use initialization $\boldsymbol{\xi}$, the Franaszek Algorithm returns a vector $\boldsymbol{x}$ such that

- either $\boldsymbol{x} = 0$ if there are no approximate eigenvectors $\leq \boldsymbol{\xi}$
- or $\boldsymbol{x} \neq 0$ that is the "largest" (component-wise) approximate eigenvector with $\boldsymbol{x} \leq \boldsymbol{\xi}$

So the initialization $\boldsymbol{\xi}$ serves as an "upper bound" (component-wise) for the returned vector.

Consistent Splitting

Definition

Let $\boldsymbol{H}$ be a labeled graph and $\boldsymbol{x} = (x_u)_{u \in V}$ be an $(\boldsymbol{A}_H, n)$ approximate eigenvector.

We do state splitting on some state u, so we get a partition of the outgoing edges $E_u = E_u^{(1)} \cup E_u^{(2)}$

The splitting is called $\boldsymbol{x}-$consistent if it "maintains" the approximate eigenvector property:

$\boldsymbol{x}$ is $(\boldsymbol{A}, n)$ approximate eigenvector means

$$\sum_{e \in E_u} x_{\tau_G(e)} \geq n x_u$$

we do state splitting

$$\sum_{e \in E_u^{(1)}} x_{\tau(e)} \geq n y^{(1)} \quad \text{and} \quad \sum_{e \in E_u^{(2)}} x_{\tau(e)} \geq n y^{(2)}$$

for some non-negative integers $y^{(1)} + y^{(2)} = x_u$

Observation

If we have a state $\boldsymbol{u}$ and some $(\boldsymbol{A}, n)$ approximate eigenvector $\boldsymbol{x}$ and we perform $\boldsymbol{x}-$consistent splitting, then we get

A new graph $\boldsymbol{H}'$ from the partition $E_u^{(1)}, E_u^{(2)}$ of E_u with two new states $u^{(1)}, u^{(2)}$

A partition of x_u into $y^{(1)}$ and $y^{(2)}$

So take a new vector $x' = (x_v)_{v \in V_{H'}}$ with

$$(x')_v = \begin{cases} x_v & \text{if } v \neq u \\ y^{(1)} & \text{if } v = u^{(1)} \\ y^{(2)} & \text{if } v = u^{(2)} \end{cases}$$

Then x' is an $(A_{H'}, n)$ approximate eigenvector.

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_u \\ \vdots \\ x_{|V|} \end{pmatrix} \longrightarrow \begin{pmatrix} x_1 \\ \vdots \\ y^{(1)} \\ y^{(2)} \\ \vdots \\ x_{|V|} \end{pmatrix} = x'$$

- We have the original Graph $\boldsymbol{H}$
- We have some $(\boldsymbol{A}_H, n)$ approximate eigenvector $\boldsymbol{x}$
- We do $\boldsymbol{x}$-**consistent splitting** and we get
  - A new graph $\boldsymbol{H}'$ with more edges
  - A new $(\boldsymbol{A}_{H'}, n)$ approximate eigenvector $\boldsymbol{x}'$ *and* that eigenvector has smaller values in its coordinates than $\boldsymbol{x}$.

Is there an $\boldsymbol{x}$-consistent splitting

If we have an irreducible graph $\boldsymbol{H}$ and $\boldsymbol{x}$ is an $\boldsymbol{A}_H$ approximate eigenvector that is not the all-one vector and is strictly positive

$\rightarrow$ Then there exists a basic $\boldsymbol{x}$-consistent splitting of $\boldsymbol{H}$

Build the Encoder

Start with a graph $\boldsymbol{G}$ of finite anticipation that describes a constraint system. Choose two integers p, q so that $p/q \leq cap(\boldsymbol{S})$.

1. Begin with $\boldsymbol{G}^q$
2. Find a $(\boldsymbol{A}_{G^q}, 2^p)$ approximate eigenvector (it exists).
   - If it is a $0-1$ a $(\boldsymbol{A}_{G^q}, 2^p)$ approximate eigenvector we are done.

Otherwise

- We reduce ourselves into the non-zero states of $\boldsymbol{x}$; let's call the subgraph $\boldsymbol{G}_0$ and the eigenvector $\boldsymbol{x}_0$:
- If the graph $\boldsymbol{G}_0$ is not irreducible, we reduce it to an irreducible sink
- We do $\boldsymbol{x}_0$-consistent splitting.
  - We get a graph $\boldsymbol{G}_1$ that is irreducible and an $(\boldsymbol{A}_{G_1}, 2^p)$ approximate eigenvector $\boldsymbol{x}_1$ with positive coordinates.
  - If that is a 0-1 vector then we are done. Otherwise repeat the same with $\boldsymbol{G}_1$ and $\boldsymbol{x}_1$

G

$$G \rightarrow \overset{G^q}{x \text{ an } (A_{G^q}, 2^p) \text{ appr. eig.}}$$

$$G \rightarrow \begin{matrix} G^q \\ x \text{ an } (A_{G^q}, 2^p) \text{ appr. eig.} \end{matrix} \xrightarrow[\text{of } x]{\text{keep non-zero entries}} \begin{matrix} G_0 \\ x_0 \end{matrix}$$

$$\boldsymbol{G} \rightarrow \begin{matrix} \boldsymbol{G}^q \\ x \text{ an } (\boldsymbol{A}_{G^q}, 2^p) \text{ appr. eig.} \end{matrix} \xrightarrow[\text{of } x]{\text{keep non-zero entries}} \begin{matrix} \boldsymbol{G}_0 \\ \boldsymbol{x}_0 \end{matrix}$$

$$\xrightarrow{\boldsymbol{x}_0 \text{ consistent splitting}} \begin{matrix} \boldsymbol{G}_1 \\ \boldsymbol{x}_1 \end{matrix}$$

$$\boldsymbol{G} \rightarrow \begin{matrix} \boldsymbol{G}^q \\ x \text{ an } (\boldsymbol{A}_{G^q}, 2^p) \text{ appr. eig.} \end{matrix} \xrightarrow[\text{of } x]{\text{keep non-zero entries}} \begin{matrix} \boldsymbol{G}_0 \\ \boldsymbol{x}_0 \end{matrix}$$

$$\xrightarrow{\boldsymbol{x}_0 \text{ consistent splitting}} \begin{matrix} \boldsymbol{G}_1 \\ \boldsymbol{x}_1 \end{matrix} \xrightarrow{\boldsymbol{x}_1 \text{ consistent splitting}} \begin{matrix} \boldsymbol{G}_2 \\ \boldsymbol{x}_2 \end{matrix}$$

$$\boldsymbol{G} \rightarrow \begin{matrix} \boldsymbol{G}^q \\ x \text{ an } (\boldsymbol{A}_{G^q}, 2^p) \text{ appr. eig.} \end{matrix} \xrightarrow[\text{of } x]{\text{keep non-zero entries}} \begin{matrix} \boldsymbol{G}_0 \\ \boldsymbol{x}_0 \end{matrix}$$

$$\xrightarrow{\boldsymbol{x}_0 \text{ consistent splitting}} \begin{matrix} \boldsymbol{G}_1 \\ \boldsymbol{x}_1 \end{matrix} \xrightarrow{\boldsymbol{x}_1 \text{ consistent splitting}} \begin{matrix} \boldsymbol{G}_2 \\ \boldsymbol{x}_2 \end{matrix}$$

$$\rightarrow \quad \ldots \quad \rightarrow \begin{matrix} \boldsymbol{G}_T \\ \boldsymbol{x}_T \end{matrix}$$

The last graph $\boldsymbol{G}_T$ has a minimum out-degree of 2^p

and out-splitting preserves finite anticipation

$\Rightarrow \boldsymbol{G}_T$ has finite anticipation

We delete excess edges and we get an $(\boldsymbol{S}^q, 2^p)$ encoder.

We tag it with input labels and we get a $p : q$ finite state encoder.

References

Chapters 1 - 5 of:

Marcus, Brian H., Ron M. Roth, and Paul H. Siegel. "An introduction to coding for constrained systems." Lecture notes (2001).