Codes on Graphs:
Generalized State Realizations

G. David Forney, Jr.

Laboratory for Information and Decision Systems
MIT
Cambridge MA 02139 USA

IMA Summer Program
6 August 1999
Outline

Review of conventional state realizations (trellises) of codes

- Codes as behavioral systems
- The concept of “state”
- Trellis sections as local constraints

Generalization: generalized state realizations

- Factor graph realizations with restrictions
- Represented by GSR graphs or normalized factor graphs
- Actually, no loss of generality
- Specifies associated decoding algorithms (sum-product algorithm)

Minimal realizations: an open problem

- Lower bound: Cut Set Bound
- Determines minimal realization in cycle-free, group case
- Examples
  - Conventional state realizations (trellises)
  - Tail-biting (single-cycle) realizations
  - Reed-Muller codes

Duality results

- Dual GSR graphs generate dual codes (group or linear).
- The sum-product algorithm may be performed on the dual graph.
Codes

A code is defined by a behavioral system $\Sigma = (I, A, C)$, where

- the *index set* (time axis) $I$ is any discrete set, not necessarily ordered;
- the *symbol sequence space* $A = \bigotimes_{k \in I} A_k$ is the Cartesian product of a collection of *symbol alphabets* $A_k, k \in I$;
- the *code* is any subset $C \subseteq A$; i.e., any set of *codewords* $a = \{a_k, k \in I\} \in A$.

**Block code:** $I$ is finite.

**Group code:** a code $C$ with a componentwise group property; i.e.,

- the symbol sequence space $A = \bigotimes_{k \in I} A_k$ is the *direct product* of a collection of *symbol groups* $A_k, k \in I$ (here, finite abelian);
- the code is a *subgroup* $C \subseteq A$.

Every linear code is a group code.
The concept of state

Based on a two-way partition of the time axis $I$ into:

- The past $k^- = \{i \in I \mid i < k\}$;
- The future $k^+ = \{i \in I \mid i \geq k\}$.

The state $s_k$ at time $k$ is any function of the past $a_{|k^-}$ such that:

- Given $s_k$ and $a_{|k^-}$, the set of possible futures $a_{|k^+}$ depends only on $s_k$; i.e.,
- Given $s_k$, the future is conditionally independent of the past (Markov property; sufficient statistic property)

Graphical representation of conditional independence:

![Graphical representation](image)

For linear or group codes:

- There is a well-defined, essentially unique minimal state space group $S_k$ at each time $k$ . . .
- . . . or in fact for any two-way partition of the index set $I$. 

Local constraints (trellis sections)

In view of the concept of state:

- The sets of possible next outputs $a_k \in A_k$ and next states $s_{k+1} \in S_{k+1}$ depend only on the current state $s_k$.

Local constraint (local code, evolution law, trellis section)

$$C_k \subseteq S_k \times A_k \times S_{k+1} :$$

set of all state transitions/outputs $(s_k, a_k, s_{k+1})$ that can possibly occur.

Graphical representation of local constraint (trellis section):

Group case:

- $C_k$ is a subgroup of the direct product group $S_k \times A_k \times S_{k+1}$; e.g., $C_k$ is a linear block code.
Conventional state realizations (trellises)

1. Establish an ordering of the index set \( I \) (the “art” of trellis design);
2. For each \( k \in I \), define a state space \( S_k \);
3. For each \( k \in I \), define a local constraint (trellis section)
   \( C_k \subseteq S_k \times A_k \times S_{k+1} \);
   • The state sequence space is the Cartesian product \( S = \bigotimes_k S_k \);
   • The configuration space is \( A \times S \);
4. The full behavior \( B \) is the set of all configurations \( (a, s) \in A \times S \) such that \( (s_{|k}, a_{|k}, s_{|k+1}) \in C_k \) for all \( k \in I \);
5. The realization generates \( C \) if the projection \( B_{\mid A} \) of the full behavior \( B \) onto the symbol sequence space \( A \) is equal to \( C \).

Graphical representation of conventional state representation (trellis):

![Trellis Diagram]

Notes:

• Every symbol vertex \( A_k \) has degree 1 and every state vertex \( S_k \) has degree 2;
• The graph is cycle-free.

Group case:

• After ordering \( I \),
  the minimal conventional state realization is completely determined.
Possible generalizations

General factor graphs (Tanner-Wiberg-Loeliger graphs)

- Bipartite graphs:
  1. Symbol and state vertices
     - Symbol vertices $A_k, k \in I_A$: given \textit{a priori}, observable;
     - State vertices $S_j, j \in I_S$: designer's choice, unobservable;
  2. Constraint vertices $C_i, i \in I_C$:
     constrain configurations of connected symbol/state vertices
     (local configuration space $\bigotimes_{k \in I_A(i)} A_k \times \bigotimes_{j \in I_S(i)} S_j$)

- The \textit{full behavior} $B$ is the set of all configurations $(a, s) \in \mathcal{A} \times \mathcal{S}$ such that all local constraints are satisfied;

- The realization \textit{generates} $C$ if the projection $B_{|\mathcal{A}}$ of the full behavior $B$ onto the symbol sequence space $\mathcal{A}$ is equal to $C$.

- Group case:
  each symbol alphabet $A_k$ and state space $S_j$ is a group, and each constraint $C_i$ is a subgroup of the direct product $\bigotimes_{k \in I_A(i)} A_k \times \bigotimes_{j \in I_S(i)} S_j$.

Trellis formations:

- Factor graphs comprised of trellis sections;
  \textit{i.e.}, every constraint vertex is connected to two state vertices.

- See next talk (Kötter and Vardy)

Generalized state realizations:

- Factor graphs restricted as follows:
  - Every symbol vertex has degree 1;
  - Every state vertex has degree 2;
  - No restrictions on constraint vertices.
GSR graphs

Generalized state realizations (factor graph picture):

- Factor graphs restricted as follows:
  - Every symbol vertex has degree 1;
  - Every state vertex has degree 2;
  - No restrictions on constraint vertices.

Generalized state realizations (GSR graph picture):

- To convert from factor graph picture:
  - Constraint vertices become ordinary vertices;
  - State vertices become ordinary edges (connecting 2 vertices);
  - Symbol vertices become “leaf edges” (connected to 1 vertex).

- Result: a “graph with leaf edges”
  - An ordinary (unipartite) graph;
  - Arbitrary graph topology;
  - Same global properties as original factor graph
    (e.g., cycle-freedom, connectedness)

- Completely equivalent, but nicer in some respects (IMHO)

System theory: a state realization, with time axis = general graph

Example:

\[
\begin{align*}
  & A_k & A_{k+1} & A_{k+2} & A_{k+3} \\
  & S_k & S_{k+1} & S_{k+2} & S_{k+3} & S_{k+4} \\
  & C_k & C_{k+1} & C_{k+2} & C_{k+3} \\
  & \ldots & & & & \ldots
\end{align*}
\]

GSR graph representation of conventional state realization (trellis)
Codes represented by factor graphs

Any factor graph may be converted to a GSR graph

Conversion (factor graph ⇒ GSR graph):

(Vertex with equals sign = repetition constraint)

An essentially identical graphical representation:
- Graph “looks the same;” topology does not change;
- No change in sum-product decoding complexity.

Application: low-density parity-check codes
- Binary symbol variables become binary state spaces.

Normalized factor graph:
- State vertices have degree 2, symbol vertices have degree 1

Any GSR graph may be converted to a normalized factor graph

Conversion (GSR graph ⇒ normalized factor graph):

Another essentially identical graphical representation:
- Graph topology does not change;
- No change in sum-product decoding complexity.

⇒ Any factor graph may be converted to a normalized factor graph
Minimal realizations: The cut set bound

Cut set: a minimal set of edges such that removal of that set partitions the graph into two disconnected subgraphs.

- A graph is cycle-free if every edge is by itself a cut set.

In a GSR graph:

- a cut set is a set of state spaces, $S_\chi = \bigotimes_{j \in \chi} S_j$;
- state = site for a cut
- fixing the values of $s_\chi \in S_\chi$ effectively removes the edges in $S_\chi$, disconnecting the graph into two independent subgraphs, and partitioning the symbol vertices into two disjoint subsets

$\Rightarrow$ Cut Set Bound: $|S_\chi| = \prod_{j \in \chi} |S_j| \geq$ minimal state space size corresponding to this two-way partition

Cycle-free case:

- GSR state space sizes are lowerbounded by conventional trellis state space sizes $\Rightarrow$ no significant complexity reduction is possible.

Group case: Once symbol variables are mapped to a cycle-free GSR graph, a canonical minimal realization is completely determined.

Cyclic case:

- Only the products of GSR state space sizes are lowerbounded $\Rightarrow$ significant complexity reductions are possible

- Minimal realizations: An open problem
  - Difficult even in the simplest single-cycle (tail-biting) case
Example: Tail-biting trellises

A tail-biting trellis is an example of a generalized state realization.

GSR graph of a tail-biting trellis:

Cut Set Bound $\Rightarrow$ Square Root Bound:

- Because a cut set involves two state spaces, minimal maximum state space size $\geq$ \textit{square root} of minimal state space size at midpoint of conventional trellis representation

- Example: (24, 12, 8) Golay code
  - Minimal conventional trellis has 256 states at midpoint;
  - There exists a minimal 16-state tail-biting trellis.
  - Decoding performance: near-ML (\(\approx 0.1\) dB loss)
**Example: Tetrahedral GSR**

Conjectured tetrahedral GSR graph for (24, 12, 8) Golay code:

- Symbol alphabets (4 leaf edges) are 64-valued (6 bits)
- State spaces (6 ordinary edges) are 4-valued
  - Cut Set Bound is met for all possible cut sets
- Constraint vertices represent (12, 6) binary linear codes

Such a GSR exists iff the Golay code has a generator matrix as follows:

```
xxxxxx   xxxx   00000   00000  
xxxxxx   xxxx   00000   00000  
xxxxxx   00000   xxxx   00000  
xxxxxx   00000   xxxx   00000  
xxxxxx   00000   00000   xxxx   
xxxxxx   00000   00000   xxxx   
000000   xxxx   xxxx   00000   
000000   xxxx   xxxx   00000   
000000   xxxx   00000   xxxx   
000000   xxxx   00000   xxxx   
000000   00000   xxxx   xxxx   
000000   00000   xxxx   xxxx   
```

Unfortunately, Vardy has shown that no such generator matrix exists.
The sum-product decoding algorithm

A generalized state realization specifies a decoding algorithm

- namely, the sum-product algorithm applied to the associated normalized factor graph.
- Sum-product algorithm: fundamentally based on cut sets
  - Cycle-free, finite: converges in finite time to exact solution;
  - Cycles: iterative, inexact.

GSR advantage: Clean functional architecture

- In a GSR graph, each graph element has a distinct function:
  1. Leaf edges (symbol variables) are for input/output;
  2. Ordinary edges (state variables) are for communication;
     (degree 2 implies pass-through, no computation);
     State complexity \(|S_j|\) = “bandwidth”
  3. All computation takes place at constraint vertices
     (only one update rule, not two)
     Constraint complexity \(|C_i|\) = “decoding complexity”

- Layering point of view
  - Symbol variables form an I/O layer;
    * “Intrinsic information” in, “extrinsic information” out
  - Constraints and state variables form a computation layer;
    * Constraint vertices = processors; state edges = interconnect

- May be iterated recursively;
  * e.g., a constraint \(C_i\) may be represented by a GSR and computed at a lower layer
Dual codes and realizations

A group code $C$ has a well-defined *dual group code* $C^\perp$

- Linear codes: usual notion of duality based on orthogonality
- Group codes: Pontryagin duality

**Dual generalized state realization** (finite abelian case):

- Same graph topology as primal GSR graph;
- Same symbol alphabets $A_k$;
- Same state spaces $S_j$;
- Replace constraint codes $C_i$ by dual codes $C_i^\perp$
  - Complexity is reduced if $|C_i| > |C_i^\perp|$ (high-rate case)
- ...and one trick (Mittelholzer) based on degree-2 property:

$$
\begin{array}{c}
\begin{array}{c}
\bigotimes
\end{array}
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
\bigoplus
\end{array}
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
S_j^1
\end{array}
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
S_j^2
\end{array}
\end{array}
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
S_j^1
\end{array}
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
S_j^2
\end{array}
\end{array}
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
S_j^1 = S_j^2
\end{array}
\end{array}
\quad
\begin{array}{c}
\begin{array}{c}
S_j^1 = -S_j^2
\end{array}
\end{array}
$$

**Theorem:** If a group GSR generates a group code $C$, then the dual GSR generates the dual group code $C^\perp$.

**Corollary** (Mittelholzer): If a group trellis generates a group code $C$, then the dual trellis generates the dual group code $C^\perp$.

Notes:

- Holds for any GSR graph, cycle-free or not.
- Dual group codes have GSRs with the same topology and state spaces
  - Dual tail-biting trellises
Parity-check and generator realizations

The dual of a parity-check (zero-sum) constraint is a repetition constraint.

The dual of a parity-check realization is a generator realization:

\[
\begin{align*}
\text{parity-check realization} & \quad \text{generator realization} \\
(n, n - r) \text{ code} & \quad (n, r) \text{ code}
\end{align*}
\]

\[
\begin{array}{c}
\text{n symbols} \\
\vdots
\end{array}
\quad
\leftrightarrow
\quad
\begin{array}{c}
\text{n symbols} \\
\vdots
\end{array}
\]

\[
\begin{array}{c}
r \text{ checks} \\
r \text{ inputs}
\end{array}
\]

**Corollary.** If \( C \) is a linear code with parity-check matrix \( H \), then the dual code \( C^\perp \) is the code whose generator matrix is \( H \).

(Note: “Low-density generator matrix” codes are not very interesting.)

Similar constructions:

- Dual kernel and image representations.
- Dual encoders and syndrome-formers.
Dualizing the sum-product algorithm

The BCJR algorithm can be performed on the trellis of the dual code (Hartmann-Rudolph [1976], Battail et al. [1979], Hagenauer et al. [1996])

Given any group code $C$ generated by any cycle-free GSR, can exact sum-product (APP) decoding be performed on the dual GSR?

Yes!

Proof: Poisson summation formula.

Method: Fourier-transform the incoming and outgoing weight vectors:

![Diagram showing the Fourier-transform process for the dual graph of the sum-product algorithm.]

Example: binary weight vector $(w_0, w_1) \equiv_\alpha (1, w_1/w_0) = (1, \lambda)$

- Fourier transform of $(1, \lambda)$ is $(1 + \lambda, 1 - \lambda) \equiv_\alpha (1, (1 - \lambda)/(1 + \lambda))$
- Leads to “tanh rule:”
  \[
  \lambda \mapsto \Lambda = \frac{1 - \lambda}{1 + \lambda}
  \]
- N.B.: range of $\lambda$ is $[0, \infty]$, but range of transform $\Lambda$ is $[-1, 1]$.

For non-binary weight vectors, Fourier transform is complex in general.
Dualizing the sum-product algorithm (cont.)

Dualization may be applied to any cycle-free graph fragment.

Example: Dualizing the sum-product update rule

1. Transform the incoming weight vectors;
2. Execute the sum-product update rule using the dual local code $C_i^\perp$;
3. Transform the resulting outgoing vector.

The dual update rule is simpler if $C_i$ is high-rate ($|C_i| > |C_i^\perp|$)

- e.g., if $C_i$ is an $(n, n-1, 2)$ parity-check constraint, then $C_i^\perp$ is an $(n, n, 1)$ repetition constraint, and the dual update rule is a simple componentwise multiplication.
Summary: Advantages of GSRs

Advantages of GSRs and GSR graphs:

- Ordinary (unipartite) graphs, with leaf edges
  - Arbitrary topology
  - Computation only at constraint vertices; only one update rule
- Clean separation of functions
  - Symbol edges = I/O
  - State edges = communication
  - Constraint vertices = computation
- State spaces are better represented by edges
  - State = communication (e.g., between “past” and “future”)
  - State = site of a possible cut
- Duality theorems
  - Dual realizations generate dual codes
  - Sum-product algorithm may be executed on the dual graph
Open questions

In general, all questions that one could ask about conventional trellises, or about, *e.g.*, tail-biting trellises. For example:

Find good GSR graph representations of known codes

- Good classes of graphs
- How best to assign symbols to vertices (the “art”)  
  - for low-complexity or “minimal” representations  
  - for best decoding performance
- Tradeoffs between state/constraint complexity and “cyclicity”
- Algebraic structure?

Find classes of GSR graphs that generate good codes and lead to good decoding performance

- *e.g.*, tail-biting trellises, turbo codes, LDPC codes

Explore modifications of sum-product decoding for graphs with cycles

...etc. It feels like early days ...