The sum-product algorithm on simple graphs

Michael. E. O'Sullivan**, John Brevik†, Shayne. M. Vargo‡
*, Department of Mathematics and Statistics
San Diego State University
San Diego, CA, USA, 92182
Email: mosulliv@math.sdsu.edu
† Department of Mathematics and Statistics
California State University,
Long Beach, CA, 90840
Email: jbrevik@csulb.edu
‡ Department of Mathematics
University of Southern California
Los Angeles, CA, 90089
Email: svargo@usc.edu

***Abstract*— This article summarizes work in progress on theoretical analysis of the sum-product algorithm. Two families of graphs with quite different characteristics are studied: graphs in which all checks have degree two and graphs with a single cycle. Each family has a relatively simple structure that allows for precise mathematical results about the convergence of the sum-product algorithm.**

I. Introduction

One of the great achievements in coding theory in the last decade was the discovery that iterative decoding methods, such as the sum-product algorithm (SPA), can be used to achieve Shannon capacity. This has been shown experimentally and proven for ensembles of codes in, for example, [2, 3.24,4.14]. Unfortunately, although there are provable asymptotic results for the performance of the sum-product algorithm, there is little that can be said for specific finite length codes. In this article we report on two simple cases for which we can derive theoretical results about convergence of the sum-product algorithm. By establishing some simple, but provable, results we hope to build a foundation for further algebraic analysis. These examples may also enhance the intuitive understanding of the algorithm and thereby yield improved heuristic methods for code construction.

We will only consider binary codes, and since the SPA is most easily described via a bipartite graph, rather than a check matrix, we will not mention the check matrix in our analysis. It has proven useful to use the following definition of a bipartite graph: A *bipartite graph* is a 5-tuple, $B = (E, L, R, \lambda, \rho)$ consisting of an edge set E and two sets, the *bit* nodes L and the *check* nodes R with two structural maps $\lambda : E \to L$ and $\rho : E \to R$ giving the ends of each edge E. A *codeword* is an association of 0 or 1 to each $\ell \in L$ such that each $r \in R$ is connected to an even number of nonzero bits. Throughout the rest of the paper $B = (E, L, R, \lambda, \rho)$ is assumed to be a connected bipartite graph.

We express all of the probabilistic data in the algorithm using "odds" ratios. The input data for bit ℓ is the odds that the actual intended or transmitted value for that bit was 1 given the signal received, that is, $u_\ell = p_\ell(1)/p_\ell(0)$. Likewise, the messages along the edges of the graph produced by the algorithm are expressed as the odds of 1. The algorithm uses the transform from the "odds of 1" domain to the "difference domain" in which a probability distribution p is represented using $p(0) - p(1)$. The function $s : \mathbb{R} \cup \{\infty\} \longrightarrow \mathbb{R} \cup \{\infty\}$ defined by $s(x) = \frac{1-x}{1+x}$ transforms from one domain to the other. Notice that $s(s(x)) = x$.

Sum-Product Algorithm
INPUT: For each $\ell \in L$, $u_\ell \in (0, \infty)$.
DATA STRUCTURES: For each $e \in E$, $x_e, y_e \in (0, \infty)$.
INITIALIZATION: Set $y_e \leftarrow 1$ for all $e \in E$.
ALGORITHM:

BIT-TO-CHECK STEP: For each $e \in E$, set

$$x_e \leftarrow u_{\lambda(e)} \prod_{\substack{f:\lambda(f)=\lambda(e) \\ f \neq e}} y_f$$

CHECK-TO-BIT STEP: For each $e \in E$, set

$$y_e \leftarrow s\left(\prod_{\substack{f:\rho(f)=\rho(e) \\ f \neq e}} s(x_f) \right)$$

NEW ESTIMATE STEP: Set

$$\hat{u}_\ell \leftarrow u_\ell \prod_{e \in \lambda^{-1}(l)} y_e$$

When necessary we indicate the iteration using a superscript

as follows. We initialize $y^{(0)} = 1$ and for $t \geq 1$,

$$x_e^{(t)} \leftarrow u_{\lambda(e)} \prod_{\substack{f:\lambda(f)=\lambda(e) \\ f \neq e}} y_f^{(t-1)}$$

$$y_e^{(t)} \leftarrow s\left(\prod_{\substack{f:\rho(f)=\rho(e) \\ f \neq e}} s(x_f^{(t)}) \right)$$

We seek conditions under which each $\hat{u}_\ell$ converges either to 0 or to ∞.

Section II treats graphs in which each check node $r \in R$ has degree two. The algorithm simplifies so that techniques from linear algebra may be applied. The final theorem describes exactly the region of convergence for the SPA. Section III treats graphs in which there is a single cycle. We introduce a slight generalization of the SPA, which allows input at the check nodes, and reduce the SPA on a single cycle graph to this generalized algorithm applied to a simple cycle. Once again we derive results using linear algebra.

II. Check nodes of degree two

It is relatively straightforward to show that if all check nodes have degree 2, then all edge messages are monomials in the u_ℓ. Furthermore, for each edge e, at any iteration t, $y_e^{(t)} = x_{\bar{e}}^{(t)}$ where $\bar{e}$ is the unique edge sharing a check node with e. Let us use $\boldsymbol{a}_e \in \mathbb{N}^L$ to denote the row vector of exponents appearing in x_e, so $x_e = \prod_{\ell \in L} u_\ell^{\boldsymbol{a}_{e,\ell}}$. We will abbreviate this product as $\boldsymbol{u}^{\boldsymbol{a}_e}$. When we want to specify the tth iteration we will write $\boldsymbol{a}_e^{(t)}$. Let $\mathbf{0} \in \mathbb{N}^L$ be the zero row vector and let $\boldsymbol{\delta}_\ell \in \mathbb{N}^L$ be the row vector which is 1 in the ℓth component and 0 otherwise. The update in the SPA is

$$x_e = \boldsymbol{u}^{\boldsymbol{a}_e} \leftarrow u_{\lambda(e)} \prod_{\substack{f:\lambda(\bar{f})=\lambda(e) \\ \bar{f} \neq e}} \boldsymbol{u}^{\boldsymbol{a}_f}$$

Keeping track of the exponents we have the following algorithm.

Local Sum Algorithm

Data Structures: For each $e \in E$, $\boldsymbol{a}_e, \in \mathbb{N}^L$.

Initialization: Set $\boldsymbol{a}_e \leftarrow \mathbf{0}$ for all $e \in E$.

Algorithm Set

$$\boldsymbol{a}_e \leftarrow \boldsymbol{\delta}_{\lambda(e)} + \sum_{\substack{f:\lambda(\bar{f})=\lambda(e) \\ \bar{f} \neq e}} \boldsymbol{a}_f \tag{1}$$

Let $\boldsymbol{A}$ be the $|E| \times |L|$ matrix whose eth row is $\boldsymbol{a}_e$ and let $\boldsymbol{\Lambda}$ be the $|E| \times |L|$ matrix with $\boldsymbol{\Lambda}_{e,l} = 1$ when $\lambda(e) = l$ and $\boldsymbol{\Lambda}_{e,l} = 0$ otherwise. So, the eth row of $\boldsymbol{\Lambda}$ is $\delta_{\lambda(e)}$. Let $\boldsymbol{K}$ be the $E \times E$ matrix $\boldsymbol{K}_{e,f} = 1$ when $\lambda(\bar{f}) = \lambda(e)$ and $\bar{f} \neq e$, otherwise $\boldsymbol{K}_{e,f} = 0$. The local sum algorithm may then be expressed as

$$\boldsymbol{A}^{(0)} = \mathbf{0} \qquad \text{and} \qquad \boldsymbol{A}^{(t)} = \boldsymbol{\Lambda} + \boldsymbol{K}\boldsymbol{A}^{(t-1)}$$

The equation above is easily solved, for $t \geq 1$,

$$\boldsymbol{A}^{(t)} = \Big(\boldsymbol{K}^{t-1} + \boldsymbol{K}^{t-2} + \cdots + \boldsymbol{K} + I \Big) \boldsymbol{\Lambda}$$

One can check that K is the adjacency matrix of a *directed* graph $\tilde{G}$, which we call the *flow graph* of B, whose *vertex set* is E. We now apply the theory of nonnegative matrices [1]. A nonnegative matrix K is called primitive when some power of the matrix is strictly positive. In this case there is a unique eigenvalue of maximum modulus, this eigenvalue is positive, and it has algebraic multiplicity one. A corresponding eigenvector, called the *Perron vector* of K, is strictly positive. The following theorem shows that the sum-product algorithm converges away from a set of measure 0 provided K is primitive.

Theorem 2.1: Suppose K is primitive. Let $\boldsymbol{y}^*$ be the left Perron vector of K and let $\boldsymbol{c} = \boldsymbol{y}^*\boldsymbol{\Lambda}$. The sum-product algorithm on B converges to zero when $\boldsymbol{u}^{\boldsymbol{c}} < 1$ and converges to 1 when $\boldsymbol{u}^{\boldsymbol{c}} > 1$.

When B is regular it follows that $\boldsymbol{y}^*$ is constant, and consequently that c is also constant. Then the SPA converges if and only if the product $\prod_{\ell \in L} u_\ell \neq 1$.

It is possible for K to not be primitive, which corresponds to $\tilde{G}$ being multipartite. Consider the simple example of two bit nodes and n check nodes, each connected to both bit nodes. Then $K^2 = \begin{bmatrix} K_1 & 0 \\ 0 & K_2 \end{bmatrix}$ with K_1 and K_2 primitive. The SPA converges if $u_1 u^{n-1}$ and $u_1^{n-1} u_2$ have the same parity, that is both larger than 1 or both smaller than 1. Otherwise the algorithm oscillates. The case $n = 3$ appears in Figure II, along with the associated flow graph, which happens to be undirected in this case. For $n = 3$

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$K^2 = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 & 0 \\ 1 & 2 & 1 & 0 & 0 & 0 \\ 1 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 1 & 1 \\ 0 & 0 & 0 & 1 & 2 & 1 \\ 0 & 0 & 0 & 1 & 1 & 2 \end{bmatrix}$$

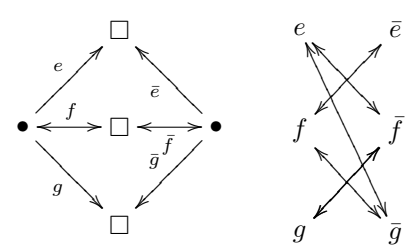

Fig. 1. A case where the SPA oscillates on a region. At left the bipartite graph B, at right the flow graph of B, $\tilde{G}$.

III. Single cycle graphs

We now assume B is a bipartite graph which has a single cycle. We may write B as the union of the cycle C with several trees, $T_1, \ldots, T_r$, each disjoint from one another and each meeting the cycle in a single node. For each tree, the messages along the edges going toward the cycle will eventually stabilize, after a number of iterations equal to the diameter of the tree. Thus after a large enough number of iterations the messages into the cycle from the trees will be constant.

A. Reduction to a cycle

We would like to compare the performance of the SPA on B, after this "settling down" period, with the performance of the SPA on C, but there are complications. First, when the messages finally stabilize, the edge messages on the cycle are no longer 1, as they where at initialization. Thus we consider more general initializations. Second, we must alter the input values at the nodes of the cycle. If one of the T_i is connected to a bit node then we may simply alter the value for the bit node by multiplying by this incoming message from T_i. In order to handle the case where one of the T_i is connected to a check node, we allow initialization of the check node. Thus we arrive at the following, slightly more general, algorithm. Note that the case $v_r = 0$ for all $r \in R$ gives the usual SPA.

Generalized Sum-Product Algorithm

INPUT : For each $l \in L$, $u_l \in (0, \infty)$.
For each $r \in R$, $v_r \in [0, \infty)$.
For each $e \in E$, $y_e^{(0)} \in (0, \infty)$.

DATA STRUCTURES: For each $e \in E$, $x_e, y_e \in (0, \infty)$.

INITIALIZATION: Set $y_e \leftarrow y_e^{(0)}$ for each $e \in E$.

BIT-TO-CHECK STEP: For each $e \in E$, set

$$x_e \leftarrow u_{\lambda(e)} \prod_{\substack{f:\lambda(f)=\lambda(e) \\ f \neq e}} y_f.$$

CHECK-TO-BIT STEP: For each $e \in E$, set

$$y_e \leftarrow s\left(s(v_{\rho(e)}) \prod_{\substack{f:\rho(f)=\rho(e) \\ f \neq e}} s(x_f) \right).$$

NEW ESTIMATE STEP:
For each $l \in L$, set

$$\hat{u}_l \leftarrow u_l \prod_{e \in \lambda^{-1}(l)} y_e.$$

For each $r \in R$, set

$$\hat{v}_r = s\left(s(v_r) \prod_{e \in \rho^{-1}(r)} s(x_e) \right).$$

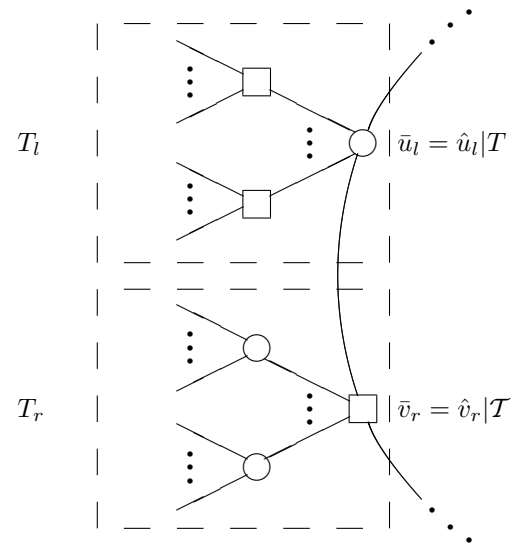

Fig. 2. Graphical representation of the reduction construction.

Again, we will declare bit l to be 0 if $\hat{u}_l < 1$, to be 1 if $\hat{u}_l > 1$, and undecided if $\hat{u}_l = 1$.

We may now reduce an instance of the GSPA on B to an instance of the GSPA on the bipartite cycle C contained in B. Let L' be the bit nodes of C and R' the check nodes of C and let T_ℓ for $\ell \in L'$ be the trees attached to the bit nodes and S_r for $r \in R'$ the trees attached to the check nodes. Any one of these may consist of a single node. Let N be some sufficient number of iterations for the incoming messages from the T_ℓ and S_r to stabilize. For each $\ell \in L'$ let $\bar{u}_\ell$ be the value to which the GSPA converges on T_ℓ at node ℓ (*i.e.* $u_\ell^{(N)}$). Similarly, let $\bar{v}_r$ be the value to which the GSPA converges on S_r at node r. For edges e of C, let $\bar{y}^{(0)} = y_e^{(N)}$ be the value that the GSPA on B attains after N iterations. We refer to the instance $\bar{u}_\ell$ for $\ell \in L'$, $\bar{v}_r$ for $r \in R$ and initialization $\bar{y}^{(0)}$ as the *reduction to C of the instance* $u_\ell, v_r, y^{(0)}$ on B.

Theorem 3.1: For each edge e of C, $\bar{x}_e^{(t)} = x_e^{(t+N)}$ and $\bar{y}_e^{(t)} = y_e^{(t+N)}$.

As a consequence of this theorem we conclude that convergences (or lack thereof), new estimates, and decisions for reduction to C coincide with those obtained on B. Thus we consider the GSPA on a simple cycle to see what may be determined about convergence.

B. The GSPA on a simple cycle

We now suppose that B is a $2r$-cycle, enumerating the bit nodes, check nodes, and edges as in Figure 3. To give us an idea of the behavior of the GSPA on the cycle, let us first consider the y update along edge 0. For iteration $t \geq r$, we

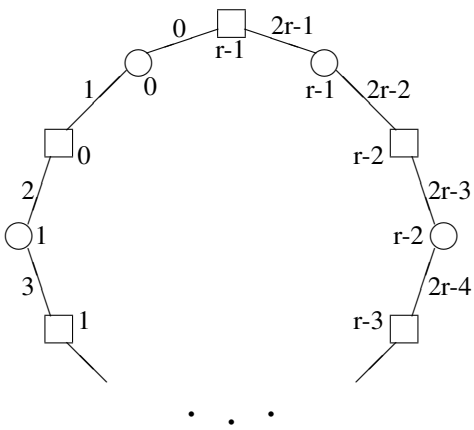

Fig. 3. Bipartite graph for the simple $\mathbf{2r}$-cycle.

obtain

$$\begin{aligned} y_0^{(t)} &= s\Big(s(v_{r-1})s\Big(x_{2r-1}^{(t)}\Big)\Big) \\ &= s\Big(s(v_{r-1})s\Big(u_{r-1}y_{2r-2}^{(t-1)}\Big)\Big) \end{aligned} \tag{2}$$

Continuing the recursive operation we could arrive at an expression for $y_0^{(t)}$ in terms of $y_0^{(t-r)}$ and the input values. It is useful to write each edge message as a fraction, $y_0 = \frac{m_0}{n_0}$ (superscripting with t as needed).

Lemma 3.2: For each $e = 0, \ldots, 2r-1$ there is a 2×2 matrix $\mathbf{M}_e$, whose entries are sums of distinct monomials in the u_ℓ, v_r, such that

$$\begin{bmatrix} m_e^{(t)} \\ n_e^{(t)} \end{bmatrix} = \mathbf{M}_e \begin{bmatrix} m_e^{(t-r)} \\ n_e^{(t-r)} \end{bmatrix}$$

In the enumeration in Figure 3, the edges 2ℓ and $2\ell+1$ are both incident on bit ℓ. The following lemma shows that the matrices $M_{2\ell}$ and for $M_{2\ell+1}$ are related. Note that these matrices correspond to clockwise versus counterclockwise flow of messages.

Lemma 3.3: Let $M_e = \begin{bmatrix} a_e & b_e \\ c_e & d_e \end{bmatrix}$. Then $a_{2\ell} = a_{2\ell+1}$, $d_{2\ell} = d_{2\ell+1}$, and $b_{2\ell}c_{2\ell} = b_{2\ell+1}c_{2\ell+1}$.

For input values $u_\ell > 0$ and $v_r \geq 0$, M_e is a strictly positive matrix, provided that at least one of the v_r is positive.

Theorem 3.4: Consider the GSPA applied to the simple $2r$-cycle with positive bit node inputs u_ℓ, non-negative check node inputs v_r and initialization $y_e^{(0)}$. If some v_r is nonzero then

$$y_e^{(\infty)} = \frac{2b_e}{d_e - a_e + \sqrt{(d_e - a_e)^2 + 4b_e c_e}}.$$

$$\hat{u}_\ell^{(\infty)} = \frac{4b_{2\ell}c_{2\ell}}{(d_{2\ell} - a_{2\ell} + \sqrt{(d_{2\ell} - a_{2\ell})^2 + 4b_{2\ell}c_{2\ell}})^2}.$$

The ℓth component of the decision vector is 0 when $a_{2k} < d_{2k}$, 1 when $a_{2k} > d_{2k}$, and undecided otherwise.

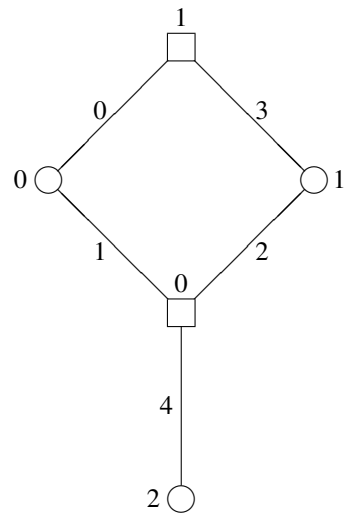

Fig. 4. Bipartite graph for the lollipop code.

C. Examples

The first example we will analyze is the Lollipop in Figure 4, a simple 4-cycle with one extraneous edge attached to a check node. The SPA will converge for all input. Let

$$\rho(u_0u_1) = \frac{\sqrt{2(u_0u_1+1)D + 6u_0^2u_1^2 - 4u_0u_1 + 6}}{4\sqrt{u_0u_1}}.$$

where $Du\sqrt{9u_0^2u_1^2 - 14u_0u_1 + 9}$. The decision vector will be

- [0,0,0], if $u_0u_1 < 1$ and $u_2 < \rho$;
- [0,0,1], if $u_0u_1 < 1$ and $u_2 > \rho$;
- [1,1,0], if $u_0u_1 > 1$ and $u_2 < \rho$;
- [1,1,1], if $u_0u_1 > 1$ and $u_2 > \rho$;
- undecided if $u_0u_1 = 1$ or $u_2 = \rho$.

This implies that the SPA my produce non-codewords. The codeword decisions occur when to $u_2 < \rho$. To better understand the behavior of ρ, Table I gives a list of values for various products of u_0u_1, along with the convergence values of the new bit estimate. Although we only show products $u_0u_1 \leq 1$, note that $\rho(1/u_0u_1) = \rho(u_0u_1)$.

D. Example 2: The Pull-Toy

Again, we can apply the reduction construction with $N = 1$. We obtain the simple 4-cycle with inputs $\bar{u}_0 = u_0$, $\bar{u}_1 = u_1$, $\bar{v}_0 = u_2$, and $\bar{v}_1 = u_3$. The initial update matrices are identical to those given in the previous example. Below, we give the convergence value for y_0 to display the growing complexity of the messages. We have

$$y_0^{(\infty)} = \frac{2(u_1u_3 + u_2)}{1 + u_1u_2u_3 - u_0u_1 - u_0u_2u_3 + \sqrt{D}},$$

where

$$D = (1+u_1u_2u_3-u_0u_1-u_0u_2u_3)^2+4(u_1u_2+u_3)(u_0u_1u_3+u_0u_2).$$

TABLE I

LIST OF VALUES PRODUCED BY THE SPA FOR THE LOLLIPOP CODE

u_0u_1	ρ	u_2	$\hat{u}_0^{(\infty)} = \hat{u}_1^{(\infty)}$	$\hat{u}_2^{(\infty)}$
0.01	8.6507	0.5	0.0025	0.0050
		1.0	0.0100	0.0200
		2.0	0.0378	0.0777
		10.0	0.3854	1.2368
0.1	2.7112	0.5	0.0291	0.0538
		1.0	0.0100	0.1980
		2.0	0.2660	0.6355
		10.0	0.7530	5.1039
0.5	1.2085	0.5	0.2679	0.3228
		1.0	0.5000	0.8000
		2.0	0.7035	1.7550
		10.0	0.9317	9.3125
1.0	1.0000	0.5	1.0000	0.5000
		1.0	1.0000	1.0000
		2.0	1.0000	2.0000
		10.0	1.0000	10.0000

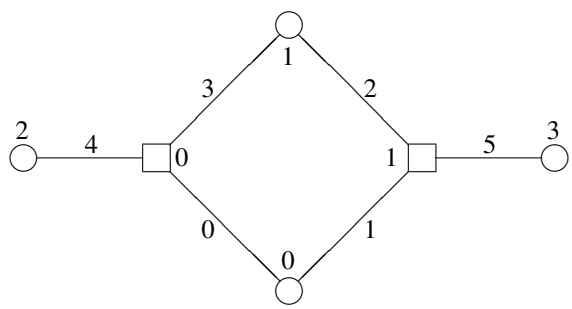

Fig. 5. Bipartite graph for the pull-toy.

Unlike the lollipop, the decisions at the bits on the cycle now depend on the exterior bit inputs. There are two different cases. The decision vector for the cycle bits will be

- [0,0] or [1,1], if $u_2u_3 < \left|\frac{1-u_0u_1}{u_0-u_1}\right|$;
- [0,1] or [1,0], if $u_2u_3 > \left|\frac{1-u_0u_1}{u_0-u_1}\right|$.

Note that while this separates codewords and non-codewords on the simple cycle, any of the four vectors might lead to a codeword (or non-codeword) for the pull-toy.

REFERENCES

[1] H. Minc, *Nonnegative Matrices*, Wiley, 1988.

[2] Richardson, T., Urbanke, R., *Modern Coding Theory*, Cambridge University Press, 2008.