مجله علمی - پژوهشی "پدافند الکترونیکی و سایبری"
سال سوم، شماره ۲، تابستان ۱۳۹۴، ص ۷۳- ۶۱

مقایسه و بهبود پیاده‌سازی الگوریتم‌های کدگشایی ترتیبی کدهای کانولوشنال

محمد هادی[1*]، محمدرضا پاکروان[2]
۱- دانشجوی دکتری مخابرات، دانشگاه صنعتی شریف
۲- دانشیار، دانشگاه صنعتی شریف
(دریافت: ۹۴/۰۱/۰۱؛ پذیرش: ۹۴/۰۷/۱۴)

چکیده

قابلیت کدهای کانولوشنال در تصحیح خطا با افزایش طول حافظه افزایش می‌یابد. اما افزایش طول حافظه، باعث افزایش پیچیدگی الگوریتم کدگشایی بهینه ویتربی می‌شود، چرا که تعداد محاسبات در الگوریتم ویتربی به‌صورت نمایی به‌طول حافظه وابسته است. افزایش پیچیدگی الگوریتم ویتربی با طول حافظه، می‌تواند پیاده‌سازی این الگوریتم را هنگام کدگشایی کدهای کانولوشنال با طول حافظه بلند ناممکن کند. به همین جهت الگوریتم‌های زیربهینه‌ای همانند Fano و Stack ارائه شده‌اند تا امکان کدگشایی ترتیبی و سریع کدهای کانولوشنال با طول حافظه بلند را فراهم کنند. در این نوشتار به معرفی شیوه‌های گوناگون پیاده‌سازی الگوریتم‌های Fano و Stack پرداخته و با ارائه نوآوری‌هایی، سرعت اجرا و حافظه مورد نیاز این الگوریتم‌ها را بهبود می‌دهیم. برای ارزیابی مزایای پیاده‌سازی‌های مختلف ارائه‌شده، آن‌ها را برمبنای قدرت تصحیح خطا، زمان کدگشایی و میزان حافظه مورد نیاز مقایسه می‌کنیم. همچنین ما از نتایج شبیه سازی استفاده می‌کنیم تا نشان دهیم که اگر وضعیت کانال مخابره زیاد خراب نباشد، الگوریتم‌های Fano و Stack می‌توانند توانایی تصحیح خطای الگوریتم بهینه ویتربی را در زمان کدگشایی بسیار کمتر، برای کدگشایی کدهای کانولوشنال با طول حافظه بلند فراهم آورند.

کلیدواژه: کدهای کانولوشنال، الگوریتم‌های کدگشایی ترتیبی کدهای کانولوشنال، الگوریتم ویتربی، الگوریتم Fano، الگوریتم Stack، طول حافظه کد کانولوشنال

۱- مقدمه

کدگذاری کانولوشنال به منظور کاهش احتمال خطا، ضمن ارسال داده روی کانال‌های مخابراتی نویزی ابداع شده است [۱]. الگوریتم کدگشایی بهینه ویتربی[1] به‌عنوان متداول‌ترین الگوریتم برای کدگشایی کدهای کانولوشنال شناخته می‌شود. الگوریتم ویتربی به‌طور گسترده در عمل استفاده می‌شود، اما از پیچیدگی محاسباتی زیاد برای کدگشایی کدهای کانولوشنال با طول حافظه (طول مقید)[2] بلند رنج می‌برد. اگرچه احتمال شکست در کدگشایی کدهای کانولوشنال به‌صورت نمایی با افزایش طول حافظه کاهش می‌یابد، پیچیدگی محاسباتی الگوریتم ویتربی برای کدگشایی کدهای کانولوشنال با طول حافظه بلند، مانع حصول عملکرد مطلوب و مورد انتظار است. امروزه الگوریتم ویتربی عموما برای کدگشایی کدهای کانولوشنال با طول حافظه کمتر از ۹ به کار می‌رود [۳- ۲].

برخلاف محدودیت مذکور برای الگوریتم ویتربی، پیچیدگی محاسباتی روش کدگشایی ترتیبی[3] از طول حافظه کد کانولوشنال مستقل است [۴]. اگرچه کدگشایی ترتیبی از نظر احتمال خطا بهینه نیست اما می‌تواند به احتمال خطای قابل قبول ضمن کدگشایی کدهای کانولوشنال با طول حافظه بلند دست یابد. در الگوریتم بهینه ویتربی برای یافتن بهترین کلمه کد، تمام حالات ممکن کلمه کدها بررسی می‌شوند این درحالی است که در کدگشایی ترتیبی تنها کلمه کدهایی که برای جواب محتمل‌ترند ارزیابی می‌گردند. نحوه گزینش و ارزیابی کلمه کدها در کدگشایی ترتیبی وابسته به سطح نویز کانال بوده و با افزایش سطح نویز می‌بایست تعداد کلمه کد بیشتری بررسی شوند [۵ و ۳- ۲].

کدگشایی ترتیبی در ابتدا توسط Wozencraft برای کدگشایی کدهای کانولوشنال مطرح شد [۷- ۶]. سپس Fano یک شیوه کدگشایی ترتیبی با بهره‌وری بسیار بالا معرفی نمود. این الگوریتم با نام الگوریتم کدگشایی ترتیبی Fano شناخته

* رایانامه نویسنده مسئول: mhadi@ee.sharif.edu

1- Optimum Viterbi Decoding Algorithm
2- Memory Length (Constraint Length)
3- Sequential Decoding

می‌شود [۸]. مطالعات Fano سرآغاز پژوهش‌های گسترده‌تر پیرامون شیوه‌های کدگشایی ترتیبی شد و در این راستا Zigangirov و Jelinek به طور مستقل الگوریتم کدگشایی ترتیبی Stack را ارائه نمودند. به الگوریتم Stack گهگاه با نام الگوریتم ZJ نیز اشاره می‌شود [۷- ۶].

ما در این نوشتار به جنبه‌های عملی پیاده‌سازی الگوریتم‌های کدگشایی ترتیبی متداول Fano و Stack می‌پردازیم. در این راستا پیاده‌سازی ساده این الگوریتم‌های در نظر گرفته و سپس با ارائه راه‌کارهایی به بهبود این پیاده‌سازی‌ها می‌پردازیم. سپس عملکرد پیاده‌سازی‌های مختلف الگوریتم‌های Fano و Stack را بر مبنای معیارهایی چون زمان کدگشایی، خطای کدگشایی و ... با یکدیگر و با الگوریتم بهینه ویتربی مقایسه می‌کنیم. راه‌اندازی الگوریتم‌های کدگشایی کدهای کانولوشنال عمدتا با یک زمان گذار اولیه همراه است. در این زمان گذار اولیه، الگوریتم کدگشایی روند مناسب برای کدگشایی را یافته و سپس عملیات کدگشایی با سرعت نسبتا یکنواخت طی می‌شود. به‌عنوان دست‌آوردی دیگر، یک ویژگی برای تعیین زمان اتمام گذار اولیه در الگوریتم‌های کدگشایی ترتیبی کدهای کانولوشنال ارائه می‌شود.

فصل دوم این مقاله به معرفی اجمالی الگوریتم‌های کدگشایی ترتیبی متداول Fano و Stack می‌پردازد. در فصل سوم به پیاده‌سازی‌های مختلف این الگوریتم‌ها اشاره‌نموده و با ارائه راه‌کارهایی عملکرد آن‌ها را بهبود می‌دهیم. فصل چهارم نیز به مقایسه عملکرد الگوریتم‌های کدگشایی ترتیبی Fano و Stack با یکدیگر و با الگوریتم ویتربی اختصاص دارد. نهایتا با یک نتیجه گیری مقاله را پایان می‌دهیم.

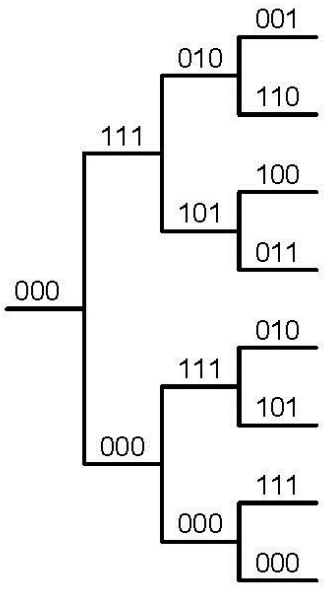
۲- الگوریتم‌های کدگشایی ترتیبی متداول

برای نمایش وابستگی در کدهای کانولوشنال می‌توان از نمودارهای حالت[1]، مشبک[2] و یا درخت[3] استفاده نمود [۳- ۲]. الگوریتم ویتربی عمدتا بر پایه نمایش مشبک توسعه می‌یابد، حال آن که الگوریتم‌های کدگشایی ترتیبی معمولا به‌وسیله نمایش درخت توصیف می‌شوند. الگوریتم‌های Fano و Stack نیز بر اساس مسیرهای موجود در نمایش درخت توصیف می‌شود. ایده اصلی در الگوریتم‌های Fano و Stack آن است که از بسط مسیرهای غیرضروری ضمن عملیات کدگشایی جلوگیری شود. ساختار الگوریتم‌های کدگشایی ترتیبی ایجاب می‌کند که ضمن اجرای الگوریتم نیازمند مقایسه مسیرهایی با طول متفاوت بر مبنای یک معیار[4] معین باشیم. معیار Fano مهم‌ترین معیار برای مقایسه رشته بیت‌های با طول متفاوت است [۸]. اگرچه میزان محاسبات در هر مرحله از الگوریتم Stack و Fano بیشتر از الگوریتم ویتربی است اما برخلاف الگوریتم ویتربی تعداد محاسبات تابعی از طول حافظه نبوده و به میزان خطای کانال بستگی دارد [۵ و ۳- ۲]. به همین جهت بسته به مقدار خطای کانال، مجموع محاسبات الگوریتم‌های Stack و Fano می‌تواند بیشتر یا کمتر از الگوریتم ویتربی باشد. هرچه میزان خطای کانال کمتر باشد، الگوریتم‌های Stack و Fano تعداد محاسبه کمتری نسبت به الگوریتم ویتربی خواهند داشت [۵ و ۳- ۲]. در ادامه ابتدا به معرفی نمایش درخت و معیار مقایسه Fano پرداخته و سپس به‌طور اجمالی الگوریتم‌های Fano و Stack را معرفی خواهیم نمود.

۲-۱- نمایش درخت

در نمایش درخت و در هر مرحله، به ازای داده‌های ورودی ممکن یک شاخه ترسیم و روی آن خروجی کدشده نظیر نوشته می‌شود. گره آغازین در نمایش درخت گره مبدا نامیده می‌شود و به قسمت انتهایی نمایش درخت که مربوط به صفر کردن حافظه است، بخش انتهایی درخت اطلاق می‌گردد. مابقی بدنه درخت نیز قسمت تقسیم کننده نام دارد. شکل(۱) قسمتی از نمایش درخت را برای کد کانولوشنال نمونه در رابطه (۱) به تصویر می‌کشد[5]. رابطه (۱) یک کد باینری با نرخ $\frac{1}{3}$ و طول حافظه 2 است [۲].

$$G(D)=\left[1+D \quad 1+D^2 \quad 1+D+D^2\right]=[6 \quad 5 \quad 7] \qquad (۱)$$



شکل (۱). قسمتی از نمایش درخت برای یک کد کانولوشنال نوعی

1- State Diagram
2- Trellis Diagram
3- Tree Diagram
4- Metric

۵- برای اطلاعات بیشتر پیرامون شیوه نمایش کدهای کانولوشنال به مراجع [۲- ۱] مراجعه شود

۲-۲- معیار مقایسه Fano

معیار فاصله اقلیدسی که مبنای مقایسه در الگوریتم ویتربی را تشکیل می‌دهد، به‌طول یکسان دو دنباله مقایسه شونده نیاز دارد. این در حالی است که در الگوریتم‌های Stack و Fano ممکن است دو دنباله با طول متفاوت مقایسه شوند. نشان داده شده است که برای یک کانال گسسته بی حافظه، بهترین ملاک برای قیاس دنباله‌های با طول متفاوت، معیار Fano است [۱۳- ۱۱]. معیار Fano برای دو بیت ارسالی و دریافتی نظیر به‌صورت زیر تعریف می‌شود:

$$M(r_l|v_l) = log_2\left(\frac{P(r_l|v_l)}{P(r_l)}\right) - R \qquad (۲)$$

که در آن $P(r_l|v_l)$ احتمال دریافت r_l به شرط ارسال v_l، $P(r_l)$ احتمال دریافت r_l و R نرخ کد است. این رابطه برای یک کانال متقارن دودویی با بیت های ورودی هم احتمال به‌صورت زیر ساده می گردد [۲]:

$$M(r_l|v_l) = \begin{cases} log_2(2p) - R, & r_l \neq v_l \\ log_2(2-2p) - R, & r_l = v_l \end{cases} \qquad (۳)$$

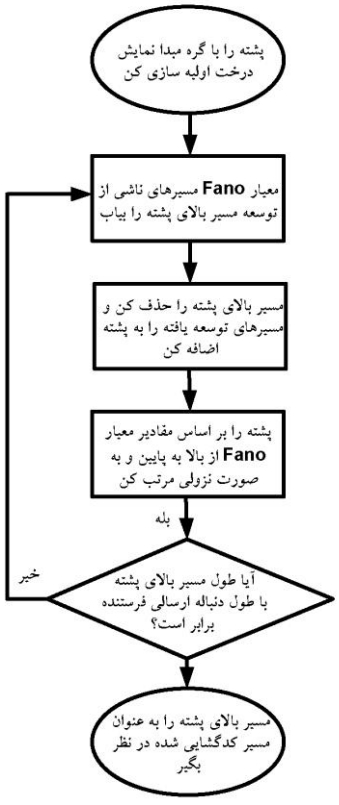
معیار Fano برای دو دنباله برابر با مجموع معیار Fano بیت‌های دو دنباله است. برای مثال برای یک کانال متقارن دودویی با احتمال خطای $p = 0.1$ و نرخ کد $R = \frac{1}{3}$ داریم:

$$M(r_l|v_l) = \begin{cases} -2.65, & r_l \neq v_l \\ +0.52, & r_l = v_l \end{cases} \qquad (۴)$$

تحت این شرایط معیار Fano دو دنباله $L_1 = [0\ 0\ 0]$ و $L_2 = [0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0\ 1\ 0\ 1\ 0\ 1\ 1]$ برابر با مقدار -1.61 است. باید توجه داشت که معیار Fano را به صورت بازگشتی نیز می توان محاسبه نمود. برای نمونه فرض کنید که دنباله L_1 با افزودن یک بیت، به دنباله $L_{ext} = [0\ 0\ 0\ 0]$ توسعه یابد. در این صورت معیار Fano دو دنباله L_{ext} و L_2 برابر با مجموع معیار Fano دو دنباله L_1 و L_2 (یعنی مقدار -1.61) و معیار Fano ناشی از توسعه مسیر (یعنی مقدار $M(0|0) = 0.52$) خواهد بود. پس معیار Fano دو دنباله L_{ext} و L_2 برابر با $-1.61 + 0.52 = -1.09$ است.

۲-۳- الگوریتم Stack

الگوریتم Stack یا ZJ از متداول‌ترین الگوریتم‌های کدگشایی ترتیبی کدهای کانولوشنال به‌شمار می‌رود. در الگوریتم Stack یک فهرست مرتب از مسیرهای با طول متفاوت کدگشایی‌شده قبلی در یک پشته[1] نگه داشته می‌شود. هر ردیف در پشته شامل مسیر کدگشایی‌شده و معیار Fano نظیر آن نسبت به دنباله دریافتی است. ردیف‌ها بر حسب معیار بیشتر از بالا به پایین مرتب شده‌اند. اگر کد به‌صورت (k, n, m) باشد، در هر گره از نمایش درخت 2^k یال خروجی داریم. در الگوریتم Stack و در هر مرحله، مسیر بالای پشته انتخاب و با افزودن داده جدید، به 2^k مسیر ممکن توسعه می یابد. برای هر مسیر حاصله معیار Fano محاسبه و نهایتا تمامی مسیرهای جدید همراه با معیارهای نظیر، جایگزین ردیف بالای پشته می‌شوند. سپس پشته بر مبنای معیار Fano از بالا به پایین به‌صورت نزولی مرتب می شود. موقعی که طول مسیر بالای پشته برابر با طول مورد انتظار دنباله کدگشایی شد، الگوریتم Stack متوقف‌شده و مسیر ردیف بالای پشته به‌عنوان مسیر کدگشایی‌شده معرفی می‌شود. الگوریتم Stack از گره مبدا در نمایش درخت و با معیارهای اولیه صفر شروع به‌کار می‌کند. نکته مهم در الگوریتم Stack آن است که توسعه پشته می‌تواند با سرعت کمتری انجام شود. در واقع در برخی حالات می‌توان مسیرهای توسعه یافته را کنار گذاشت و وارد پشته ننمود. برای مثال برای یک کد کانولوشنال دودویی، اگر معیارهای Fano برای دو مسیر توسعه‌یافته، از بزرگترین معیار Fano موجود در پشته کمتر باشند تنها کافیست مسیر توسعه‌یافته با معیار Fano بیشتر را جایگزین مسیر بالای پشته نمود. به بیان دیگر در این حالت مسیر توسعه‌یافته با معیار Fano کمتر را می‌توان حذف کرد. شکل (۲) فلوچارت الگوریتم Stack را نمایش می دهد [۲ و ۵].



شکل (۲). فلوچارت الگوریتم Stack

1- Stack

۲-۴- الگوریتم Fano

الگوریتم Fano یکی از مشهورترین الگوریتم‌های کدگشایی ترتیبی است که به پشته برای ذخیره داده‌ها نیاز ندارد. در نمایش درخت، تمامی مسیرهای کد از یکدیگر جدا هستند. از آن جا که الگوریتم Fano امکان شناسایی و تفکیک مسیرهای کد ادغام شده را ندارد، نمایش درخت مبنای مناسبی برای عملکرد آن خواهد بود. در هر مرحله، الگوریتم Fano اطلاعات سه مسیر فعلی[1]، بعدی[2] و قبلی[3] را نگاه می‌دارد. بر مبنای این اطلاعات، الگوریتم Fano می‌تواند از مسیر فعلی به مسیر بعدی یا مسیر قبلی نقل مکان کند و از این جهت نیازی به پشته برای ذخیره اطلاعات مسیرهای بررسی شده ندارد.

حرکت الگوریتم Fano توسط یک آستانه پویا[4] که مضرب صحیحی از یک مقدار گام[5] ثابت است، کنترل می‌شود. در الگوریتم Fano تنها مسیری که معیار Fano آن نسبت به دنباله دریافتی کمتر از آستانه نیست، در مرحله بعد ملاقات می‌شود. بر مبنای الگوریتم Fano، مادامی که معیار Fano مسیر بعدی از آستانه بیشتر است، فرایند توسعه مسیر ادامه می‌یابد. اگر در خلال الگوریتم، معیار Fano تمامی مسیرهای بعدی از آستانه کمتر باشد، الگوریتم به مسیر قبلی بازمی گردد، مشروط بر آن که معیار Fano مسیر قبلی از مقدار آستانه بیشتر باشد. در غیر این صورت آستانه به میزان یک گام کاهش می‌یابد تا الگوریتم در مسیر فعلی به دام نیفتد. در الگوریتم Fano، اگر یک مسیر دو بار بازبینی شود، مقدار آستانه در مرتبه دوم از آستانه در مرتبه اول کمتر خواهد بود. این امر سبب می‌شود در اجرای الگوریتم حلقه بسته ایجاد نشده و نهایتا الگوریتم همگرا شود. شکل (۳) فلوچارت الگوریتم Fano را نمایش می‌دهد. در شکل (۳)، دنباله داده، معیار Fano، آستانه و گام به ترتیب با v، M، T و Δ نشان داده شده‌اند. زیرنویس‌های c، p، s و t نیز مبین مسیرهای فعلی، قبلی، بعدی و موقتی هستند.

در هر مرحله از اجرای الگوریتم Fano یکی از پنج اقدام ممکن اتخاذ می‌شود. این اقدام‌ها شامل حرکت به جلو همراه با تنگ کردن آستانه[6]، حرکت به جلو[7]، کاهش آستانه[8]، حرکت به عقب به‌صورت موفقیت‌آمیز[9] و حرکت به عقب به‌صورت غیرموفقیت آمیز[10] می‌باشند. پیاده‌سازی‌های مختلف الگوریتم‌های Fano و Stack و ارائه راه‌کارهایی برای بهبود آن‌ها در این بخش به پیاده‌سازی الگوریتم‌های Stack و Fano می‌پردازیم. سپس با ارائه راهکارهایی، پیاده‌سازی‌ها را مرتبه به مرتبه بهبود می‌دهیم. بهبودهای معرفی شده در راستای شیوه پیاده‌سازی این الگوریتم‌ها بوده و به تغییر ساختار خود الگوریتم‌ها مربوط نمی‌شود. به علاوه، در توضیحات مربوط به هر نسخه پیاده‌سازی شده، به کلیات و ایده‌های نهفته در پیاده‌سازی اشاره شده و از تبیین جزئیات که عمدتا شیوه کدنویسی را شامل می‌شوند، خودداری می‌شود. همچنین در توضیحات آتی به منظور تسهیل در بیان مطالب، فرض می‌کنیم با کدهای کانولوشنال دودویی با نرخ کد $\frac{1}{n}$ کار می کنیم. تعمیم مباحث به کدهای کانولوشنال دلخواه به سادگی ممکن است. زیربخش اول از این قسمت مقاله، به پیاده‌سازی الگوریتم Stack و زیربخش دوم به پیاده‌سازی الگوریتم Fano اختصاص داده شده است.

۳-۱- پیاده‌سازی‌های متفاوت الگوریتم Stack

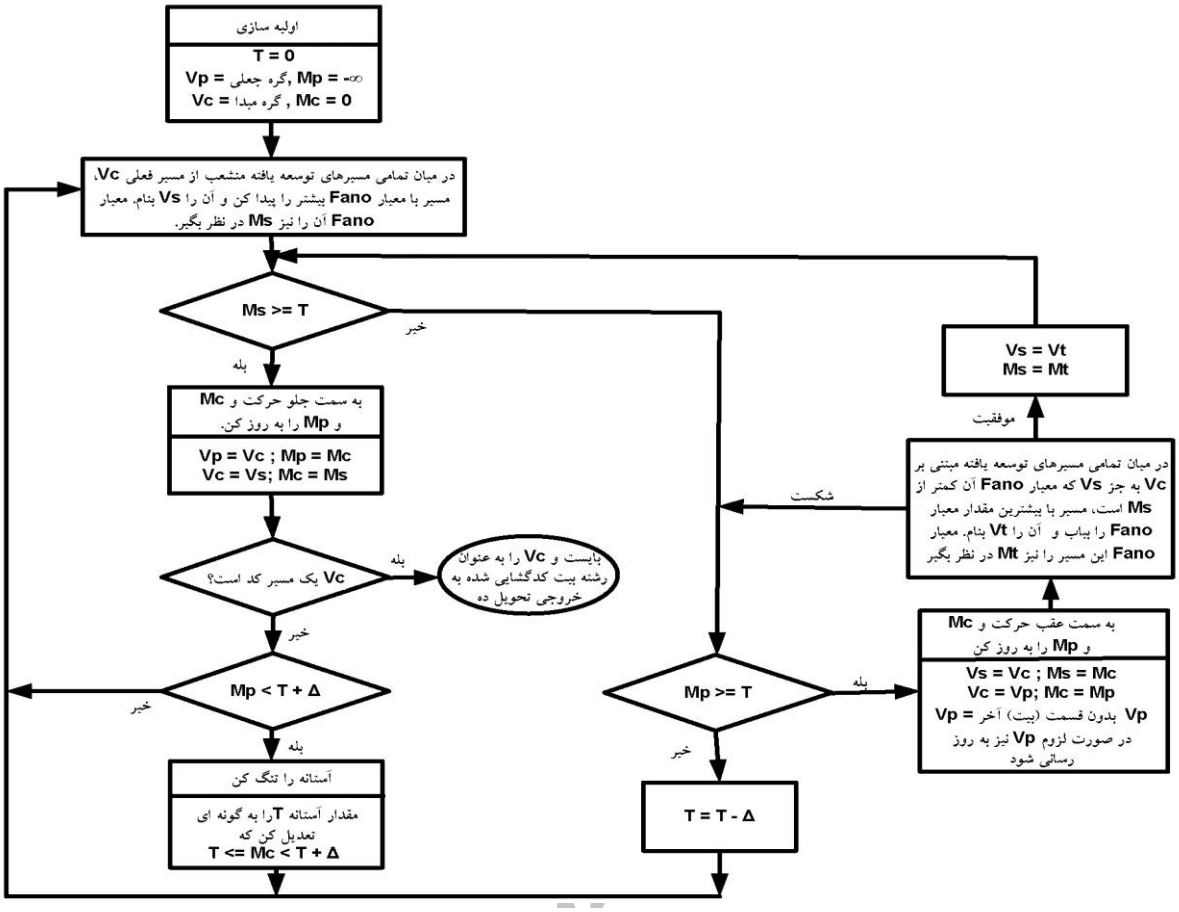
فرایند پیاده‌سازی الگوریتم Stack می‌تواند به چندین پیاده‌سازی اولیه تقسیم شود. هر مرحله از الگوریتم Stack شامل توسعه ردیف بالای پشته و محاسبه خروجی کدکننده به‌ازای مسیرهای توسعه‌یافته یا معادلا پیشروی روی نمایش درخت، محاسبه معیار Fano و مرتب‌سازی پشته است. با یک دیدگاه سلسله مراتبی، می‌توان از توابعی مجزا برای محاسبه معیار Fano یا کدکننده کانولوشنال استفاده کرد. به‌علاوه ممکن است یک تابع برای تبدیل شیوه معمول نمایش چندجمله‌ای‌های توصیف‌کننده کد کانولوشنال، به‌شیوه مناسب برای پیاده‌سازی (مثلا نمایش دودویی) نیز مورد نیاز باشد. در ادامه، از تبیین جزئیات پیاده‌سازی سلسله مراتبی الگوریتم صرف‌نظر نموده و به راه‌کارها و ایده‌های مورد استفاده در نسخه‌های متفاوت پیاده‌سازی اشاره می‌نماییم.

۳-۱-۱- نسخه اول پیاده‌سازی الگوریتم Stack

هدف از این پیاده‌سازی اولیه و ساده، ایجاد یک بستر برای توسعه ایده‌های آتی است. از این‌رو در این پیاده‌سازی، هر مرحله از الگوریتم Stack بدون هیچ‌گونه نوآوری پیاده‌سازی می‌شود. رشته بیت کدشده دریافتی در گیرنده، ماتریس مولد کد و خطای کانال؛ به‌عنوان ورودی به پیاده‌سازی نسخه اول وارد می‌شوند. خروجی این پیاده‌سازی، رشته بیت کدگشایی‌شده بر مبنای الگوریتم Stack است.

در پیاده‌سازی نسخه اول، رشته بیت دریافتی در گیرنده به بخش‌هایی به‌طول تعداد بیت خروجی کدکننده کانولوشنال

1- Current Path
2- Successor Path
3- Predecessor Path
4- Dynamic Threshold
5- Step
6- Move Forward Tighten Threshold(MFTT)
7- Move Forward (MF)
8- Lower Threshold (LT)
9- Move Backward Successfully (MBS)
10- Move Backward with Fail (MBF)



شکل (۳). فلوچارت الگوریتم Fano

تقسیم می‌شود. در واقع اگر نرخ کد کانولوشنال $R = \frac{1}{n}$ باشد، خروجی به قسمت‌های n بیتی تقسیم می شود. دقت شود فرض می‌شود که در این تقسیم‌بندی رشته بیت دریافتی گیرنده و رشته بیت ارسالی فرستنده همزمان هستند. برای درک مفهوم همزمانی فرض کنید که رشته بیت 110 به کدکننده کانولوشنال متداول [131 171]. اعمال می شود. در این صورت رشته بیت خروجی برابر با 110101 خواهد بود که در آن 11 نظیر بیت ورودی 1، 01، نظیر بیت ورودی 1 و 01 نظیر بیت ورودی صفر است. اکنون تفکیک رشته بیت کدشده به‌صورت 11\01\01 یک تفکیک همزمان و تفکیک مطابق با 1\10\10\1 یک تفکیک ناهمزمان است. الگوریتم Stack با صفر کردن محتوای پشته و سایر متغیرها شروع به کار می کند. در هر مرحله، مسیر بالای پشته با افزودن دو بیت صفر و یک به دو مسیر جدید توسعه می‌یابد. این دو مسیر وارد کدکننده کانولوشنال شده و رشته بیت کدشده نظیر هر کدام به‌دست می‌آید. برای هریک از این رشته بیت‌های کدشده، معیار Fano نسبت به رشته بیت دریافتی گیرنده محاسبه و در پشته ذخیره می‌شود. سپس پشته بر مبنای معیار Fano و به‌صورت نزولی از بالا به پایین مرتب می‌شود. این مراحل تا وقتی که تعداد بیت مسیر بالای پشته با تعدا بیت کدشده در فرستنده برابر شود، ادامه می‌یابد.

در این پیاده‌سازی، تمامی مسیرهای توسعه‌یافته همراه با معیار Fano و خروجی کدشده نظیر در پشته ذخیره می‌شوند. به علاوه محدودیتی روی حجم پشته قرار نمی‌دهیم. مسلما با افزایش حجم پشته، عملیات مرتب‌سازی طولانی و چالش برانگیز می‌شود. در عین حال این پیاده‌سازی می‌تواند برای نمایش جزئیات عملکردی الگوریتم Stack به‌کار رود.

۳-۱-۲- نسخه دوم پیاده‌سازی الگوریتم Stack

به‌طور عادی با هر مرحله پیشروی الگوریتم Stack، اندازه پشته یک واحد افزایش می‌یابد. پیش از این اشاره شد که اگر در الگوریتم Stack، معیار Fano برای تمام مسیرهای توسعه یافته از حداکثر معیار Fano موجود در پشته کمتر باشد، می‌توان مسیر توسعه یافته با کم‌ترین معیار Fano را نادیده گرفت. استفاده از این نکته باعث می‌شود که برای کدهای کانولوشنال دودویی و درصورت برقراری شرایط در یک مرحله نوعی از الگوریتم، اندازه پشته افزایش نیابد. این ویژگی در پیاده‌سازی نسخه دوم الگوریتم Stack لحاظ می‌شود.

چنانچه اشاره شد معیار Fano را می‌توان به‌صورت بازگشتی محاسبه نمود، به این معنی که معیار Fano در یک مرحله برابر با معیار Fano در مرحله قبلی به اضافه مقدار جزئی ناشی از

توسعه مسیر است. این موضوع علاوه بر کاهش بار محاسباتی معیار Fano، نیاز به محاسبه تمامی مسیر کدشده را به‌ازای هر مسیر موجود در پشته، برطرف می‌سازد. در واقع در هر مرحله باید چند بیت کدشده ناشی از توسعه مسیر بالای پشته را یافت. سپس معیار Fano این چند بیت کدشده را نسبت به بیت‌های نظیر در رشته بیت کدشده دریافتی محاسبه نموده و به مقدار پیشین معیار Fano افزود. در پیاده‌سازی نسخه دوم الگوریتم Stack، معیار Fano به‌صورت بازگشتی محاسبه و از ذخیره بی‌مورد رشته بیت خروجی خودداری می‌شود.

توجه به دو نکته مذکور سبب می‌شود که پیاده‌سازی نسخه دوم دارای اندازه پشته کمتر چه از منظر عمق و چه از منظر عرض پشته باشد. مسلما کاهش عمق پشته، افزایش سرعت مرتب‌سازی پشته را به‌همراه دارد. افزون بر این، کاهش اندازه پشته در دو بعد، حافظه مورد نیاز برای اجرای الگوریتم را کمتر می‌کند.

۳-۱-۳- نسخه سوم پیاده‌سازی الگوریتم Stack

بر مبنای مشاهدات تجربی می‌توان دریافت که مسیرهای ذخیره‌شده در بالای پشته صرف‌نظر از قسمت‌های انتهایی مسیرها، یکسانند. از دیگر سو، هرچه به قسمت انتهایی پشته نزدیک می‌شویم، احتمال آن‌که مسیر به‌عنوان رشته بیت کدگشایی‌شده برگردانده شود کمتر می‌شود [۲ و ۵]. این موضوع این نکته را به ذهن متبادر می‌کند که می‌توان در هر مرحله قسمت مشترک مسیرهای موجود در پشته را به‌عنوان قسمتی از مسیر کدگشایی شده حذف و در نتیجه اندازه پشته را کاهش داد. این ویژگی در شرایطی که کانال خطای کمتری داشته باشد، نمود بیشتری خواهد داشت. شناسایی قسمت مشترک مسیرها و ارسال آن به خروجی به عنوان قسمتی از مسیر کدگشایی شده، ضمن کاهش حجم حافظه پشته مورد نیاز، در زمان بسیار کوتاهی (با پیچیدگی خطی نسبت به تعداد مسیرهای موجود در پشته) قابل انجام است. به‌علاوه در شرایط مناسب کانال، این موضوع خطای زیاد در کدگشایی پدید نمی‌آورد. در پیاده‌سازی نسخه سوم، این نکته را مد نظر قرارداده و به‌صورت پویا ضمن اجرای الگوریتم، قسمت مشترک مسیرهای موجود در پشته را به‌عنوان قسمتی از رشته بیت کدگشایی‌شده به خروجی منتقل می‌کنیم. در این پیاده‌سازی اندازه عمق پشته را نیز محدود و به‌عنوان یک پارامتر ورودی در نظر می‌گیریم تا بتوانیم تعداد معین از مسیرهای کم ارزش انتهای پشته را دور بریزیم (در یک راه‌کار پیچیده‌تر می‌توان مسیرهای موجود در پشته را بر مبنای میزان فاصله اقلیدسی معیار Fano آن‌ها از معیار Fano مسیر ابتدای پشته، ارزش‌گذاری نموده و یک مسیر مستعد حذف از پشته را یافت). بر مبنای این نکات، حجم حافظه مورد نیاز برای اجرای نسخه سوم الگوریتم Stack بسیار کاهش‌یافته و به‌صورت پویا ضمن اجرا تغییر می‌کند. کاهش حجم حافظه به‌طور ضمنی افزایش سرعت اجرای الگوریتم را به‌همراه خواهد داشت.

۳-۱-۴-نسخه چهارم پیاده‌سازی الگوریتم Stack

اگر شرایط کانال به اندازه کافی مناسب باشد می‌توان عرض پشته برای ذخیره مسیرهای کدگشایی‌شده را نیز به‌عنوان پارامتر ورودی دریافت و به اندازه دلخواه محدود کرد. در واقع، اگر شرایط کانال از منظر خطا مناسب باشد (خطا کم باشد) پشته عمدتا از مسیرهای مشابه به هم پر می شود که معیار Fano آن‌ها نسبت به معیار Fano مسیر متناظر با رشته بیت ارسالی، اختلاف ناچیز دارند. در این‌حالت تفاوت مسیرهای موجود در پشته تنها در قسمت‌های انتهایی مسیرها می‌باشد. از این‌رو می‌توان تقریبا فرض نمود که مسیرها حداکثر در تعداد بیت معینی تفاوت دارند و در نتیجه می‌توان قسمتی از مسیرها که مستعد تفاوت هستند را در پشته نگه داشت و قسمت مشترک را به‌عنوان خروجی کدگشایی شده در نظر گرفت. شکل‌های (۴ و ۵) گویای این موضوع هستند. در این پیاده‌سازی بر روی حداکثر تعداد بیت متفاوت مسیرها حد معینی در نظر گرفته و عرض پشته را محدود می‌کنیم. اگرچه این امر می‌تواند دقت عملیات کدگشایی را تا حدی کاهش دهد اما به دلیل کاهش حجم پشته، می‌تواند افزایش سرعت زیاد به‌همراه داشته باشد.

1.40	1	1	0	1	0	1
1.35	1	1	0	1	0	0
1.32	1	1	0	0	0	0
1.27	1	1	0	1	1	1
1.21	1	1	0	0	0	1
1.11	1	1	0	0	1	1

شکل (۴). یک پشته نوعی که در آن معیار Fano نظیر هر مسیر در سمت چپ آن نمایش داده شده است. با توجه به خطای کم کانال، پشته از مسیرهای مشابه پر شده است. این امر از نزدیک بودن معیارهای Fano مشهود است.

1.40	1	0	1
1.35	1	0	0
1.32	0	0	0
1.27	1	1	1
1.21	0	0	1
1.11	0	1	1

شکل (۵): شهودا می‌توان فرض کرد که تفاوت مسیرها در سه بیت انتهایی ان‌ها است. پس می‌توان سه بیت انتهایی هر مسیر را نگاه داشت و قسمت مشترک مسیرها را به‌عنوان بخشی از رشته بیت ارسالی به خروجی فرستاد. با این کار عرض پشته به سه بیت محدود شده است.

۳-۱-۵- نسخه پنجم پیاده‌سازی الگوریتم Stack

عملیات مرتب‌سازی پشته، یکی از قسمت‌های زمان بر در اجرای هر مرحله از الگوریتم Stack است. برای کاهش میزان بار محاسباتی مرتب‌سازی پشته، گونه‌ای از الگوریتم Stack با نام الگوریتم Stack-Bucket ارائه شده است [۱۵ -۱۴]. در این الگوریتم محدوده تغییرات معیار Fano مسیرهای ممکن به تعدادی بازه با اندازه معین تقسیم‌شده و به هر بازه یک فضای ذخیره داده تحت عنوان سطل[1] اختصاص می‌یابد. در طول عملیات کدگشایی، مسیری که در بالای اولین سطل غیرتهی قرار دارد توسعه می‌یابد. سپس معیار Fano برای مسیرهای توسعه یافته محاسبه‌شده و هر مسیر در سطلی که معیار Fano مسیر را شامل می‌شود، قرار می‌گیرد. نهایتا، سطل‌هایی که مسیرهای توسعه یافته جدید را شامل می‌شوند، بر مبنای اندازه معیار Fano مرتب می‌شوند. در الگوریتم Stack-Bucket بار محاسباتی از اندازه عمق پشته مستقل و به تعداد سطل‌ها بستگی دارد و به‌همین جهت سرعت اجرای این الگوریتم از الگوریتم Stack ساده بیشتر است. تقریبا اکثر پیاده‌سازی‌های نرم افزاری الگوریتم Stack، از شیوه پیشنهادی در الگوریتم Stack-Bucket استفاده می‌کنند. در عین حال باید توجه داشت که در هر مرحله از الگوریتم Stack-Bucket ممکن است مسیر با بهترین معیار Fano برای توسعه انتخاب نشود و در نتیجه عملکرد از منظر خطا تا حدی افت کند[۵ و ۱۵-۱۴]. پیاده‌سازی نسخه پنجم الگوریتم Stack، مبتنی بر شیوه Stack-Bucket است.پیاده‌سازی Stack-Bucket را می‌توان به نوآوری‌های سایر نسخ نیز تجهیز نمود تا یک الگوریتم جامع‌تر و سریع‌تر به‌دست آید.

۳-۱-۶- نسخه ششم پیاده‌سازی الگوریتم Stack

در پیاده‌سازی‌های پیشین تمام رشته بیت کدشده دریافتی اتخاذ و سپس فرایند کدگشایی ترتیبی روی آن اعمال می‌گردد. در کدگشایی بی‌درنگ تمام رشته بیت دریافتی در یک زمان معین در اختیار نیست. به‌علاوه ممکن است در شرایطی حجم رشته بیت دریافتی به‌اندازه‌ای زیاد باشد که الگوریتم توانایی مدیریت لحظه‌ای آن را نداشته باشد. در چنین شرایطی می‌توان رشته بیت دریافتی را به بخش‌هایی با طول معین تقسیم نمود. الگوریتم Stack متوالیا به هریک از این بخش‌ها اعمال‌شده و به‌منظور حفظ توالی در کدگشایی، متغیرهای معینی ضمن گذار از بخش‌ها تبادل می‌شوند. مقادیر معیار Fano مسیرهای موجود در پشته از جمله متغیرهایی است که باید ضمن گذار الگوریتم تبادل شوند. به این شیوه پیاده‌سازی که در آن متغیرهای معینی در یک حلقه بسته به الگوریتم وارد و سپس خارج می‌شوند، پیاده‌سازی حلقه‌ای[2] الگوریتم Stack اطلاق می‌کنیم. هر یک از پیاده‌سازی‌های معرفی‌شده برای الگوریتم Stack می‌تواند به‌صورت حلقه‌ای نیز پیاده‌سازی شود.

۳-۲- پیاده‌سازی‌های متفاوت الگوریتم Fano

مشابه با الگوریتم Stack، فرایند پیاده‌سازی الگوریتم Fano نیز می‌تواند به چندین پیاده‌سازی اولیه تقسیم شود. مجددا با یک دیدگاه سلسله مراتبی، می‌توان از توابعی مجزا برای محاسبه معیار Fano، کدکننده کانولوشنال و یا تبدیل شیوه معمول نمایش چندجمله‌ای‌های توصیف‌کننده کد کانولوشنال استفاده کرد. در ادامه، از تبیین جزئیات پیاده‌سازی سلسله مراتبی الگوریتم صرف‌نظر نموده و به راه‌کارها و ایده‌های مورد استفاده در نسخه‌های متفاوت پیاده‌سازی اشاره می‌نماییم.

۳-۲-۱- نسخه اول پیاده‌سازی الگوریتم Fano

در این پیاده‌سازی، تمامی جوانب الگوریتم Fano را در نظر گرفته و یک روند جامع و عاری از هرگونه بهبود پیش می‌گیریم. پیشروی الگوریتم Fano مبتنی بر سه مسیر فعلی، قبلی و بعدی است. در پیاده‌سازی نسخه اول برای هر یک از این مسیرها یک بردار حافظه در نظر گرفته می‌شود. ورودی‌های الگوریتم در این پیاده‌سازی، رشته بیت کدشده دریافتی، ماتریس مولد کد، احتمال خطای کانال و طول گام الگوریتم Fano هستند. رشته بیت کدگشایی‌شده نیز خروجی الگوریتم پیاده‌سازی شده است. در هر مرحله از الگوریتم، مسیر فعلی با افزودن بیت‌های 0 و 1 به دو مسیر جدید توسعه می‌یابد. بر مبنای مقادیر معیارهای Fano و آستانه پویا، الگوریتم یکی از پنج اقدام ممکن حرکت به جلو همراه با تنگ کردن آستانه، حرکت به جلو، کاهش آستانه، حرکت به عقب‌به‌صورت موفقیت‌آمیز و حرکت به عقب‌به‌صورت غیرموفقیت‌آمیز را اتخاذ می‌کند. این روند تا جایی ادامه می‌یابد که طول مسیر جاری برابر با طول دنباله ارسالی در فرستنده شود. به‌منظور درک شهودی عملکرد الگوریتم، اقدام‌های اتخاذی در هر مرحله می‌تواند ثبت شوند.

1- Bucket

2- Loop

پیشروی الگوریتم پیاده‌سازی‌شده شامل دو قسمت قفل کردن[1] و کدگشایی می‌باشد. فرایند قفل کردن مبین زمان گذار اولیه‌ای است که پیش از این به آن اشاره شد. در بسیاری از مواقع، ابتدای رشته بیت کدشده ارسالی فرستنده، برای گیرنده مشخص نیست. به بیان دیگر، گیرنده به یک نسخه شیفت‌یافته از داده کدشده ارسالی دسترسی دارد. خوشبختانه عملکرد کدهای روانه‌ای[2] همچون کد کانولوشنال، به این شیفت حساس نیست. الگوریتم Fano به‌عنوان یک الگوریتم کدگشایی روانه‌ای نیز به این شیفت وابسته نیست. در واقع در قسمت نخست پیشروی الگوریتم یعنی قفل کردن، الگوریتم Fano تلاش می‌کند تا مسیر مناسب را بر مبنای داده کدشده شیفت‌یافته استخراج نماید. زمان فرایند قفل کردن به میزان خطای کانال و مقدار شیفت بستگی دارد و پس از اتمام آن، الگوریتم Fano وارد قسمت عادی کدگشایی شده و به‌ازای هر مرحله یک بیت کدگشایی می‌کند. زمان اتمام قفل کردن را می‌توان بر اساس بررسی ویژگی‌های معینی از الگوریتم تعیین نمود. یک معیار مناسب می‌تواند نحوه رفتار معیار Fano مسیر جاری باشد. نتایج تجربی و نظری نشان می‌دهند که پس از اتمام قفل کردن، معیار Fano مسیر اصلی، تقریبا روند صعودی دارد. ما از این ویژگی برای شناسایی زمان اتمام فرایند قفل کردن استفاده می‌کنیم.

باید توجه داشت که الگوریتم Stack نیز از شیفت ممکن در رشته بیت کدشده ارسالی مستقل است اما در قیاس با الگوریتم Fano، عملکرد بدتری برای مقابله با شیفت موجود در رشته بیت کدشده دارد. از این رو فرایند قفل کردن یا همان گذار اولیه در الگوریتم Stack عموما زمان بیشتری نیاز دارد.

۳-۲-۲- نسخه دوم پیاده‌سازی الگوریتم Fano

اگرچه الگوریتم Fano بر مبنای سه مسیر فعلی، قبلی و بعدی عمل می‌کند لیکن لزومی به ذخیره‌سازی تمامی این مسیرها نیست. در واقع تنها کافی است مسیر فعلی را ذخیره نمود و مسیرهای بعدی و قبلی را بر مبنای آن تعیین کرد. برای این منظور می‌توان یک آرایه برداری در نظر گرفت و مکان بیت مورد نظر در مسیر فعلی را با یک اندیس نمایش داد. افزودن و یا کاستن یک واحد از این اندیس، مکان بیت مورد نظر در مسیرهای بعدی و قبلی را تعیین می‌نماید. شکل (۶) مفهوم این پیاده‌سازی مبتنی بر اندیس را نشان می‌دهد. در این پیاده‌سازی از ثبت اقدامات اتخاذی هر مرحله از الگوریتم نیز اجتناب می‌شود. با پیشروی الگوریتم Fano، مقادیر بیت‌های ابتدای مسیر فعلی تقریبا ثابت می‌شوند و از این رو می‌توان قسمت ابتدایی مسیر فعلی را به‌عنوان رشته بیت کدگشایی‌شده به خروجی منتقل نمود. چنان که در قسمت نتایج خواهیم دید، به‌کارگیری این نکات باعث می‌شود تا سرعت و حافظه مورد نیاز در پیاده‌سازی نسخه دوم الگوریتم Fano تا حد قابل توجهی بهبود یابد.

شکل (۶). مفهوم پیاده‌سازی اندیسی الگوریتم Fano

۳-۲-۳- نسخه سوم پیاده‌سازی الگوریتم Fano

در پیاده‌سازی‌های پیشین تمام رشته بیت کدشده دریافتی اتخاذ و سپس فرایند کدگشایی ترتیبی روی آن اعمال می‌گردد. این درحالی است که در کدگشایی بی درنگ تمام رشته بیت دریافتی در یک زمان معین در اختیار نیست. به‌علاوه ممکن است در شرایطی حجم رشته بیت دریافتی به اندازه‌ای زیاد باشد که الگوریتم توانایی مدیریت لحظه‌ای آن را نداشته باشد. در چنین شرایطی می‌توان رشته بیت دریافتی را به بخش‌هایی با طول معین تقسیم نمود. سپس الگوریتم Fano را متوالیا به هر یک از این بخش‌ها اعمال کرده و به‌منظور حفظ توالی در کدگشایی، متغیرهای معینی را ضمن گذار از بخش‌ها تبادل نمود. مقادیر معیار Fano مسیرهای فعلی، قبلی و بعدی از جمله متغیرهایی هستند که باید ضمن گذار الگوریتم تبادل شوند. به این شیوه پیاده‌سازی که در آن متغیرهای معینی در یک حلقه بسته به الگوریتم وارد و سپس خارج می‌شوند، پیاده‌سازی حلقه‌ای الگوریتم Fano اطلاق می‌کنیم. هر یک از دو پیاده‌سازی معرفی‌شده برای الگوریتم Fano می‌تواند به‌صورت حلقه‌ای پیاده‌سازی شود.

۳-۳- نوآوری‌ها و ابتکارات به‌کارگرفته‌شده در پیاده‌سازی‌های مختلف

هریک از نسخه‌های متفاوت پیاده‌سازی‌شده برای الگوریتم‌های Fano و Stack، دارای نوآوری‌های خاص خود است. اگرچه ممکن است برخی از این نوآوری‌ها در یک پیاده‌سازی نظامی یا تجاری تعبیه شده باشد ولی در مقالات و مآخذ گزارشی پیرامون آن‌ها موجود نیست. اصولا پیاده‌سازی‌های مرتبط با الگوریتم‌های حساس و کاربردی همچون الگوریتم‌های Stack و Fano، جنبه نظامی و یا تجاری داشته و جزئیات پیاده‌سازی‌های آن‌ها منتشر نمی‌گردد.

1- Lock
2- Stream Codes

جدول (۱). نوآوری‌های به‌کارگرفته‌شده در پیاده‌سازی‌های مختلف انجام‌شده

الگوریتم	شماره نسخه	اهم نوآوری‌ها
الگوریتم Stack	نسخه اول	✓ نسخه مرجع پیاده‌سازی الگوریتم Stack که برای مقایسه در شبیه‌سازی‌های مقاله به‌کار گرفته شده است.
	نسخه دوم	✓ محاسبه بازگشتی معیار Fano ✓ عدم توسعه پشته اگر معیار Fano تمام مسیرهای توسعه یافته از حداکثر معیار Fano موجود در پشته کمتر باشد. ✓ عدم ذخیره رشته‌بیت‌های اضافی مانند رشته‌بیت کدشده نظیر هر مسیر
	نسخه سوم	✓ محدود کردن عمق پشته با تحمیل مقدار عمق پشته ✓ محدود کردن عرض پشته با حذف قسمت مشترک مسیرهای پشته
	نسخه چهارم	✓ محدود کردن عمق پشته با تحمیل مقدار عمق پشته ✓ محدود کردن عرض پشته با تحمیل مقدار عرض پشته
	نسخه پنجم	✓ پیاده‌سازی بومی الگوریتم Stack-Bucket ✓ تجهیز پیاده‌سازی Stack-Bucket به هریک از ابتکارات گنجانده‌شده در پیاده‌سازی‌های پیشین
	نسخه ششم	✓ پیاده‌سازی حلقوی الگوریتم Stack ✓ ارتقای پیاده‌سازی‌های سایر نسخ با پیاده‌سازی حلقوی
الگوریتم Fano	نسخه اول	✓ نسخه مرجع پیاده‌سازی الگوریتم Fano که برای مقایسه در شبیه‌سازی‌های مقاله به‌کار گرفته شده است. ✓ ارائه یک ویژگی برای شناسایی زمان قفل کردن
	نسخه دوم	✓ عدم ذخیره‌سازی رشته‌بیت‌های اضافی ✓ پیاده‌سازی اندیسی الگوریتم Fano ✓ عدم نگه‌داری قسمت مشترک مسیر فعلی
	نسخه سوم	✓ پیاده‌سازی حلقوی الگوریتم Fano ✓ ارتقای پیاده‌سازی‌های سایر نسخ با پیاده‌سازی حلقوی

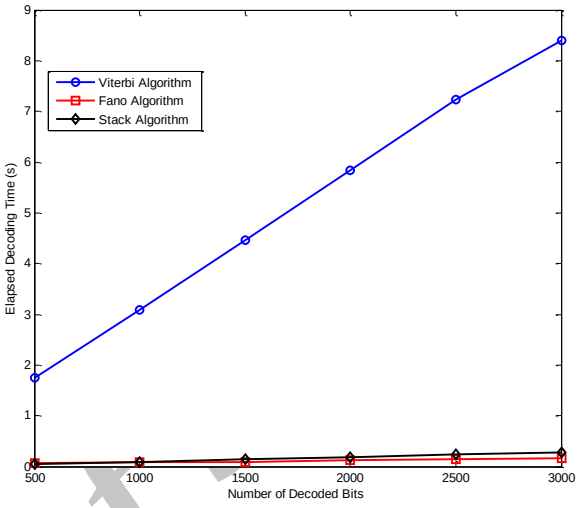
به هر روی حتی اگر ایده‌های به‌کارگرفته‌شده در پیاده‌سازی‌های این نوشتار در یک پیاده‌سازی تجاری یا نظامی نیز تعبیه شده باشد، از منظر بومی بودن می‌توان آن‌ها را نوآوری تلقی نمود. جدول (۱) خلاصه‌ای از اهم نوآوری‌ها را به تفکیک نسخه پیاده‌سازی در برمی‌گیرد. جزئیات بیشتر مربوط به هر نوآوری، در توضیحات مرتبط با نسخه پیاده‌سازی قابل دست‌یابی است.

۴- نتایج شبیه‌سازی

مبنای مقایسه و ارزیابی عملکرد یک پیاده‌سازی، قیاس عملکرد آن نسبت یک پیاده‌سازی مرجع است. در این نوشتار پیاده‌سازی‌های نسخه اول الگوریتم‌های Fano و Stack، نقش مبنای مقایسه را بر عهده دارند. در واقع، پیاده‌سازی‌های نسخه اول، همان پیاده‌سازی‌های ساده و متداول در کتب کدینگ می‌باشند که عمدتا سنگ محک نویسندگان برای کارها و ایده‌های جدید هستند. در این قسمت عملکرد الگوریتم‌های Stack، Fano و ویتربی را بر مبنای ملاک‌هایی همچون زمان کدگشایی، خطای کدگشایی و بر اساس نتایج شبیه‌سازی مقایسه می‌کنیم. پیاده‌سازی‌ها در نرم افزار MATLAB انجام و سپس بر روی یک رایانه با مشخصات Core i5 CPU و 4GB RAM اجرا می‌شوند.

شکل (۷) زمان لازم برای کدگشایی یک رشته بیت کدشده با کد کانولوشنال [1111001 1011011] را بر حسب تعداد بیت کدگشایی‌شده توسط الگوریتم‌های Stack، Fano و ویتربی نشان می‌دهد. کد کانولوشنال [1111001 1011011] یک کد کانولوشنال با طول حافظه 19 بوده و در نتیجه یک کد با طول حافظه بلند محسوب می‌شود. هم چنان که انتظار داشتیم و شکل (۷) نیز نشان می‌دهد، زمان کدگشایی الگوریتم‌های Fano و Stack به‌ازای طول ثابت دنباله کدگشایی‌شده، به‌مراتب از الگوریتم ویتربی کمتر است، درحالی‌که زمان کدگشایی الگوریتم‌های Fano و Stack تفاوت چندانی ندارند. دقت شود که این بهبود در زمان کدگشایی، با کاهش طول حافظه کد کم‌رنگ‌تر خواهد شد به‌گونه‌ای که برای کدهای کانولوشنال با

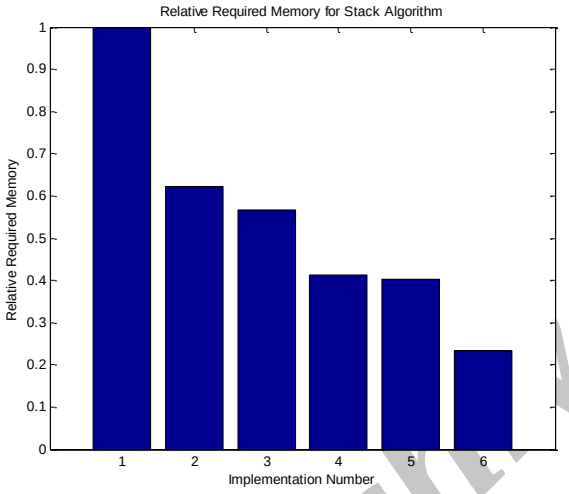
طول حافظه کم، الگوریتم ویتربی زمان کدگشایی کمتری خواهد داشت. نکته قابل توجه دیگر آن است که زمان کدگشایی سه الگوریتم رابطه‌ای تقریبا خطی با طول دنباله کدگشایی‌شده داشته و شیب خط برای الگوریتم‌های Fano و Stack از ویتربی کمتر است.



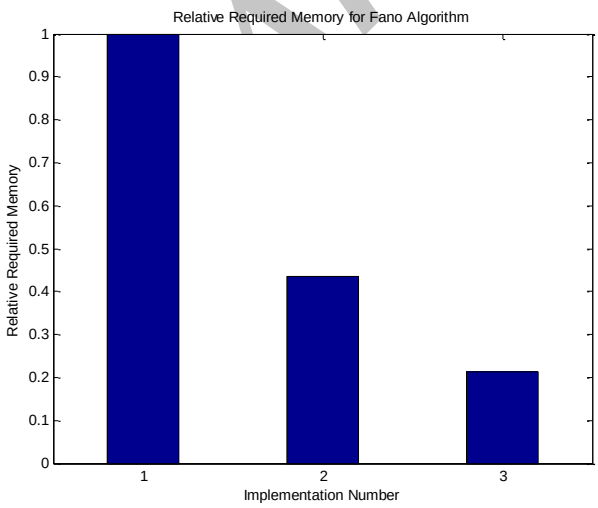
شکل (۷). زمان کدگشایی الگوریتم‌های Fano، Stack و ویتربی برای کدگشایی یک داده کدشده با کد کانولوشنال با طول حافظه بلند بر حسب طول دنباله داده کدگشایی شده

پیش از این اشاره شد که نسخه‌های متفاوت پیاده‌سازی‌شده در زمان کدگشایی و میزان حافظه مورد نیاز تفاوت دارند. برای نمونه، پیاده‌سازی نسخه سوم الگوریتم Stack دارای مدیریت حافظه پویا بوده و نسبت به نسخه‌های اول و دوم حافظه کمتری برای اجرا نیاز دارد. از طرف دیگر، نسخه دوم پیاده‌سازی الگوریتم Fano از منظر حافظه نسبت به نسخه اول برتری دارد. اگرچه پیرامون حجم حافظه مورد نیاز اظهار نظر دقیقی نمی‌توان ارائه کرد ولی قیاس نسبی میزان حافظه مورد نیاز بر مبنای شبیه‌سازی ممکن است. شکل (۸ و ۹) میزان حافظه مورد نیاز نسخه‌های متفاوت الگوریتم‌های Stack و Fano را نسبت به حافظه مورد نیاز نسخه‌های پیاده‌سازی اول الگوریتم‌ها نشان می‌دهد. این مقایسه حاصل انجام شبیه‌سازی نسخه‌های پیاده‌سازی مختلف برای رشته‌بیت‌های گوناگون و در شرایط متفاوت کانال است و به‌طور میانگین و نسبی، متوسط حافظه مورد نیاز را به‌دست می‌دهد.

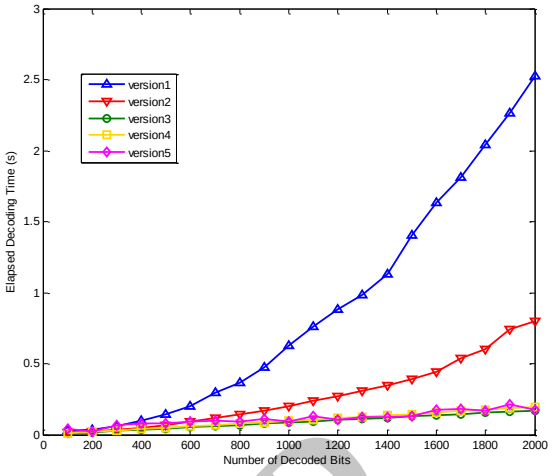
برای مقایسه زمان کدگشایی نسخه‌های متفاوت الگوریتم‌های Fano و Stack به شکل‌های (۱۰ و ۱۱) توجه نمایید. در این تصاویر زمان کدگشایی یک رشته بیت کدشده با کد کانولوشنال [1111001 1011011] بر حسب طول دنباله کدگشایی‌شده برای نسخه‌های متفاوت پیاده‌سازی‌شده، ترسیم شده است. از آن جا که پیاده‌سازی حلقه‌ای می‌تواند مبتنی بر هر یک از نسخه‌های دیگر پیاده‌سازی باشد، در مقایسه زمان کدگشایی در نظر گرفته نشده است. همچنان که شکل (۱۰) نشان می‌دهد، پیاده‌سازی‌های سوم، چهارم و پنجم الگوریتم Stack زمان کدگشایی تقریبا یکسان و بسیار کمتر از زمان کدگشایی نسخه‌های اول و دوم ارائه می‌دهند. در نقطه مقابل، پیاده‌سازی اول بیشترین زمان کدگشایی را دارد. در خصوص الگوریتم Fano نیز، پیاده‌سازی نسخه دوم از نظر زمان کدگشایی عملکرد بهتر دارد. دقت شود که رفتار تمامی نسخه‌های پیاده‌سازی‌شده با طول دنباله کدگشایی‌شده تقریبا خطی است، اگرچه در نسخه‌های اول و دوم پیاده‌سازی الگوریتم Stack که عملیات مرتب‌سازی پشته زمان قابل توجهی نیاز دارد، این رفتار تا حدی غیرخطی شده است.



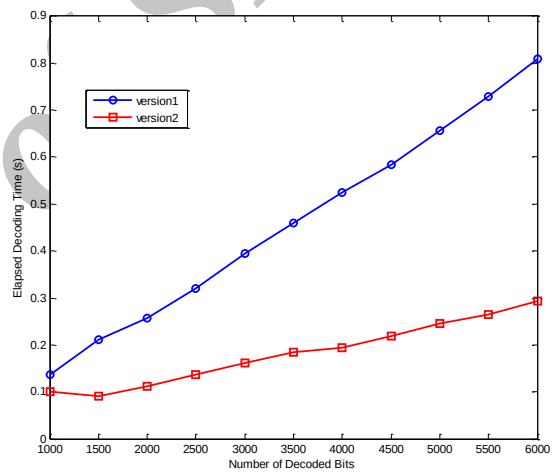
شکل (۸). میزان نسبی حافظه مورد نیاز در پیاده‌سازی‌های متفاوت الگوریتم Stack



شکل (۹). میزان نسبی حافظه مورد نیاز در پیاده‌سازی‌های متفاوت الگوریتم Fano



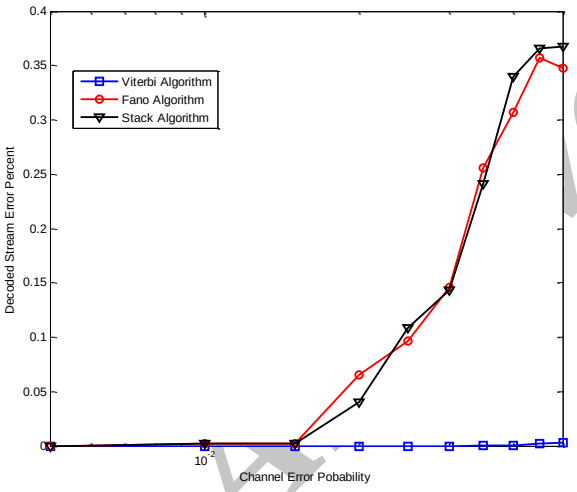
شکل (۱۰): زمان کدگشایی نسخه‌های متفاوت پیاده‌سازی الگوریتم Stack برای کدگشایی یک داده کدشده نوعی با کد کانولوشنال بر حسب طول دنباله داده کدگشایی‌شده



شکل (۱۱). زمان کدگشایی نسخه‌های متفاوت پیاده‌سازی الگوریتم Fano برای کدگشایی یک داده کدشده نوعی با کد کانولوشنال بر حسب طول دنباله داده کدگشایی‌شده

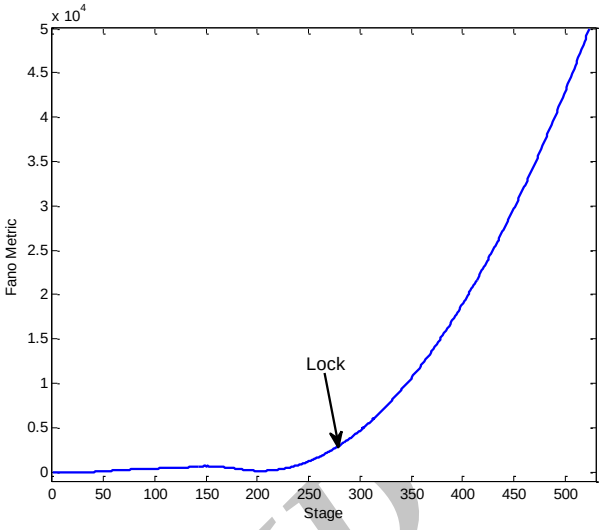
الگوریتم‌های Fano و Stack در مقایسه با الگوریتم ویتربی توانایی کمتری در تصحیح خطای ایجادشده در کانال دارند. این موضوع در شکل (۱۲) نشان داده شده است. در این شکل درصد خطای دنباله کدگشایی شده بر حسب خطای کانال برای الگوریتم‌های Fano (پیاده‌سازی نسخه دوم)، Stack (پیاده‌سازی نسخه سوم) و ویتربی مقایسه شده‌اند. رشته بیت فرستنده با کد کانولوشنال [1111001 1011011] کد شده است. به‌وضوح با افزایش میزان خطای کانال، توانایی تصحیح الگوریتم‌های Fano و Stack به شدت کاهش یافته است. نکته مهم آن است که در خطای کانال اندک، سه الگوریتم عملکرد یکسانی دارند، منتها الگوریتم‌های Stack و Fano این عملکرد را در زمان کدگشایی بسیار کمتر ارائه می‌کند. این همان مصالحه ضمنی بین دقت در کدگشایی (خطای کدگشایی) و زمان کدگشایی است که در الگوریتم‌های کدگشایی ترتیبی نمود می‌یابد. نکته مهم دیگر آن است که الگوریتم Fano، کدگشایی سریع را بدون نیاز به حافظه برای پشته انجام می‌دهد. باید توجه داشت که الگوریتم‌های Fano و Stack از منظر خطای کدگشایی تفاوت چندانی ندارند.

الگوریتم‌های کدگشایی ترتیبی می‌توانند داده ارسالی را از نسخه شیفت‌یافته داده کدشده، استخراج نمایند. بررسی‌های تجربی نشان می‌دهد که الگوریتم Fano نسبت به الگوریتم Stack، توانایی بیشتری در استخراج داده ارسالی از نسخه شیفت‌یافته دارد. هنگامی که داده کدشده شیفت‌یافته به الگوریتم‌های کدگشایی ترتیبی اعمال شود، مدت زمانی طول می‌کشد تا الگوریتم روال عادی کدگشایی را آغاز کند. ما از این زمان به زمان قفل کردن تعبیر نمودیم. انتهای عملیات قفل کردن را می‌توان بر مبنای ویژگی معینی یافت. بررسی‌های ما نشان می‌دهد که نحوه تغییر معیار Fano، می‌تواند در تعیین زمان انتهای قفل کردن راه گشا باشد.



شکل (۱۲). درصد خطای موجود در رشته بیت کدگشایی‌شده بر حسب میزان خطای کانال هنگام استفاده از الگوریتم‌های Stack، Fano و ویتربی برای کدگشایی یک رشته بیت کدشده نوعی با کد کانولوشنال

شکل (۱۳) معیار Fano مسیر فعلی را در الگوریتم Fano و برای داده کدشده شیفت‌یافته با کد کانولوشنال [1111001 1011011] بر حسب مرحله الگوریتم نشان می‌دهد. همان‌گونه که مشاهده می‌شود معیار Fano در ابتدا پیشروی الگوریتم رفتار نوسانی دارد اما پس از مدتی و به‌طور ناگهانی رفتار صعودی می‌یابد. نقطه‌ای که پس از آن معیار Fano رفتار تقریبا صعودی دارد می‌تواند مبین اتمام زمان قفل کردن باشد.



شکل (۱۳). معیار Fano برای مسیر فعلی در هر مرحله اجرای الگوریتم Fano روی داده کدشده شیفت‌یافته با کد کانولوشنال [**1111001 1011011**]

۵- نتیجه‌گیری

الگوریتم ویتربی به‌عنوان الگوریتم کدگشایی بهینه، عملا از کدگشایی کدهای کانولوشنال با طول حافظه بلند در زمان معقول ناتوان است. این درحالی است که الگوریتم‌های زیربهینه کدگشایی ترتیبی می‌توانند عملیات کدگشایی کدهای کانولوشنال با طول حافظه بلند را در زمان بسیار کوتاه انجام‌داده و در عین حال از منظر خطای کدگشایی با الگوریتم ویتربی تفاوت چندانی نداشته باشند. در این نوشتار با دیدگاهی عملی، به پیاده‌سازی الگوریتم‌های Stack و Fano به‌عنوان متداول‌ترین الگوریتم‌های کدگشایی ترتیبی پرداختیم. در این راستا نحوه عملکرد این الگوریتم‌ها را با معرفی عناوینی چون نمایش درخت و معیار Fano، به‌صورت خلاصه تشریح نمودیم. در ادامه بدون تبیین جزییات پیاده‌سازی، راه‌کارها و ایده‌هایی برای بهبود پیاده‌سازی الگوریتم‌های Fano و Stack ارائه کردیم. با به‌کار گیری این ایده‌ها، نسخه‌های پیاده‌سازی متفاوتی از این الگوریتم‌ها معرفی‌کرده و سپس عملکرد آن‌ها را بر اساس معیارهایی چون زمان کدگشایی و خطای کدگشایی ارزیابی نمودیم. نتایج شبیه‌سازی نشان دادند که الگوریتم‌های Fano و Stack در مقایسه با الگوریتم ویتربی می‌توانند در زمان بسیار کوتاه‌تر یک رشته بیت کدشده با کد کانولوشنال با طول حافظه بلند را کدگشایی کنند. همچنین اگر شرایط کانال مناسب باشد، الگوریتم‌های Fano، Stack و ویتربی از منظر خطای کدگشایی عملکرد تقریبا یکسانی دارند. الگوریتم‌های کدگشایی کدهای کانولوشنال می‌توانند داده ارسالی را از نسخه شیفت‌یافته داده کدشده نیز استخراج کنند. از این منظر عملکرد الگوریتم‌های Fano و Stack دارای یک گذار اولیه هستند. ما این گذار اولیه را قفل کردن نامیده و معیاری برای تشخیص زمان اتمام فرایند قفل کردن معرفی کردیم. این معیار مبتنی بر رفتار معیار مقایسه Fano ضمن اجرای الگوریتم‌های Fano و Stack بود.

۶- مراجع

[1] A. J. Viterbi, "Convolutional Codes and Their Performance in Communication Systems," IEEE Transaction on Communication Technology, vol. COM-19, no. 5, pp. 751-772, Oct. 1971.

[2] S. Lin and D. J. Costello, "Error Control Coding," Prentice Hall, 2004.

[3] S. Kim and S. B. Wicker, "Fundamentals of Codes Graphs and Iterative Decoding," Springer, 2002.

[4] J. B. Anderson, "Sequential Coding Algorithms: A Survey and Cost Analysis," IEEE Transaction on Communications, vol. COM-32, no. 2, pp. 169-176, Feb. 1984.

[5] S. H. Yunghsiang and C. Po-Ning, "Sequential Decoding of Convolutional Codes," Wiley Encyclopedia of Telecommunications, Apr. 2003, Revised Jul. 2015.

[6] J. M. Wozencraft, "Sequential Decoding for Reliable Communication," IRE Nat. Conv. Rec., vol. 5, no. 2, pp. 11-25, 1957.

[7] J. M. Wozencraft and B. Reiffen, "Sequential Decoding," IEEE Trans. Inform. Theory, vol. IT-9, no. 2, pp. 64-73, April 1961.

[8] R. M. Fano, "A Heuristic Discussion of Probabilistic Decoding," IEEE Trans. Inform. Theory, vol. IT-9, no. 2, pp. 64-73, April 1963.

[9] F. Jelinek, "A Fast Sequential Decoding Algorithm Using a Stack," IBM J. Res. and Dev., vol. 13, pp. 675-685, November 1989.

[10] K. S. Zigangirov, "Some Sequential Decoding Procedures," Probl. Peredachi Inf., vol. 2, pp. 13-25, 1966.

[11] J. L. Massey, "Variable Length Codes and The Fano Metric," IEEE Trans. Inform. Theory, vol. IT-18, no. 1, pp. 196-198, January 1972.

[12] E. A. Bucher, "Sequential Decoding of Systematic and Nonsystematic Convolutional Codes with Arbitrary Decoder Bias," IEEE Trans. Inform. Theory, vol. IT-16, no. 5, pp. 611-624, September 1970.

[13] F. Jelinek, "Upper bound on sequential decoding performance parameters," IEEE Trans. Inform. Theory, vol. IT-20, no. 2, pp. 227-239, March 1974.

[14] D. Haccoun and M. J. Ferguson, "Generalized Stack Algorithms for Decoding Convolutional Codes," IEEE Trans. Inform. Theory, vol. IT-21, no. 6, pp. 638-651, November 1975.

[15] J. M. Geist, "An Empirical Comparison of Two Sequential Decoding Algorithms," IEEE Trans. Commun. Technol., vol. COM-19, no. 4, pp. 415-419, August 1971.

Journal Of Electronical & Cyber Defence
Vol. 3, No. 2, 2015, Serial No. 10

Sequential Decoding Algorithms of Convolutional Codes: Implementation, Improvement and Comparison

M. Hadi*, M. R. Pakravan
Sharif University of Technology
(Received: 21/03/2015, Accepted: 06/10/2015)

ABSTRACT

Error correction capability of convolutional codes is improved by increasing code constraint length. However, increasing the constraint length results in high complexity of optimum Viterbi decoding algorithm because the number of computations in Viterbi algorithm is exponentially proportional to the constraint length. Consequently, decoding of high constraint length convolutional codes using Viterbi algorithm may practically be impossible. Sub-optimum decoding algorithms such as Fano and Stack algorithms have been proposed to feasible fast sequential decoding of high constraint length convolutional codes. In this paper, we introduce different methods of implementing Fano and Stack algorithms and propose some techniques to improve their speed and required memory. We compare the introduced implementations of the algorithms in terms of error correction capability, decoding time and required memory. Furthermore, we use simulation results to show that if the communication error is low, Fano and Stack algorithms can provide the same error correction capability as the optimum Viterbi algorithm in very short decoding time.

Keywords: Convolutional Codes, Sequential Decoding of Convolutional Codes, Viterbi Algorithm, Fano Algorithm, Stack Algorithm, Convolutional Code Constraint length.

* Corresponding Author Email: mhadi@ee.sharif.edu