Serial-Turbo-Trellis-Coded Modulation With Rate-1 Inner Code

Coders and decoders for bandwidth- and power-limited systems could be less complex.

NASA's Jet Propulsion Laboratory, Pasadena, California

Serially concatenated turbo codes have been proposed to satisfy requirements for low bit- and word-error rates and for low (in comparison with related previous codes) complexity of coding and decoding algorithms and thus low complexity of coding and decoding circuitry. These codes are applicable to such high-level modulations as octonary phase-shift keying (8PSK) and 16-state quadrature amplitude modulation (16QAM); the signal product obtained by applying one of these codes to one of these modulations is denoted, generally, as "serially concatenated trellis-coded modulation" ("SCTCM"). These codes could be particularly beneficial for communication systems that must be designed and operated subject to limitations on bandwidth and power.

Some background information is prerequisite to a meaningful summary of this development. Trellis-coded modulation (TCM) is now a well-established technique in digital communications. A turbo code combines binary component codes (which typically include trellis codes) with interleaving. A turbo code of the type that has been studied prior to this development is composed of parallel concatenated convolutional codes (PCCCs) implemented by two or more constituent systematic encoders joined through one or more interleavers. The input information bits feed the first encoder and, after having been scrambled by the interleaver, enter the second encoder. A code word of a parallel concatenated code consists of the input bits to the first encoder followed by the parity check bits of both encoders. The suboptimal iterative decoding structure for such a code is modular, and consists of a set of concatenated decoding modules — one for each constituent code — connected through an interleaver identical to the one in the encoder side. Each decoder performs weighted soft decoding of the input sequence. PCCCs yield very large coding gains at the cost of a reduction in the data rate and/or an increase in bandwidth.

As its full name suggests, SCTCM merges serially concatenated convolutional codes (SCCCs) with TCM. SCTCM is believed to offer the potential to achieve low bit-error rates ($\leq 10^{-9}$), in part because the error floors of SCCCs are lower than those of PCCCs.

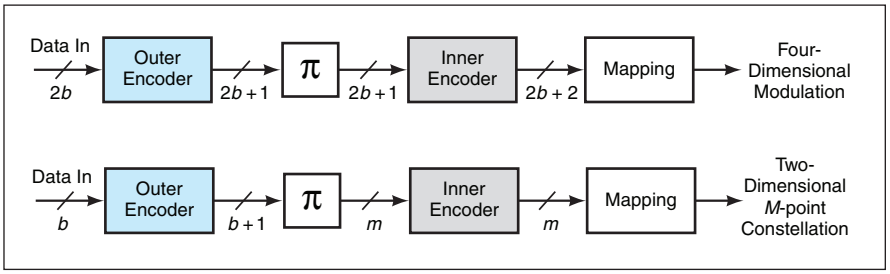
It is important to note that the proposed serial concatenated coding scheme differs from "classical" concatenated coding schemes. In a classical scheme, the role of the interleaver between two encoders is merely to break up bursts of errors produced by the inner decoder, and no attempt is made to consider the combination of the two encoders and the interleaver as a single entity. In SCTCM, on the other hand, one seeks to optimize the whole serial structure.

No attempt at such optimization was made in the past, in part because optimizing an overall code with large deterministic interleavers is prohibitively complex. However, by introducing the concept of a uniform interleaver, it is possible to draw some criteria to optimize the component codes for the construction of powerful serially concatenated codes with large block size. Another reason optimization of overall codes was not attempted is that optimum decoding of complex codes is practically impossible; only the use of suboptimum iterative decoding techniques makes it possible to decode such complex codes. The decoder in an SCTCM system would utilize an adapted version of iterative decoding used in PCCC schemes.

The upper part of the figure is a block diagram of an encoder in an SCTCM system that yields a bit-rate-to-bandwidth ratio of b (bits/second)/Hertz when ideal Nyquist pulse shaping is used. The outer encoder implements a rate-$[2b/(2b+1)]$ binary convolutional code (or a short block code) with maximum free Hamming distance (or minimum distance). The interleaver (π) permutes the output of the outer encoder. The interleaved data enter the inner encoder, which implements a rate-$[(2b+1)/(2b+2)]$ recursive convolutional code. The $2b+2$ output bits are then mapped to two symbols, each belonging to a 2^{b+1}-point two-dimensional constellation. This results in four-dimensional modulation. In this way, $2b$ information bits are used for every two modulation symbol intervals; in other words, there are b information bits per modulation symbol. The inner code and the mapping are jointly optimized on the basis of maximizing the effective free Euclidean distance of the inner TCM.

Unfortunately, the decoder associated with such an encoder would be unacceptably complex and thus unsuitable for high-speed operation. This is because the number of transitions per state for the inner TCM is 2^{2b+1} and so the number of edges in the trellis section of the decoder would have to equal to 2^{2b+1} × the number of states.

The lower part of the figure is a block diagram of an SCTCM encoder for an M-point two-dimensional constellation that would enable the use of a decoder of lower complexity. This encoder yields a bit-rate-to-bandwidth ratio of $bm/(b+1)$ (bits/second)/Hertz [where $m \equiv \log_2 M$ and M is the number of points in a two-dimensional signal constellation] when ideal Nyquist pulse shaping is used. The outer encoder implements a rate-



These **Encoders** implement two SCTCM schemes. Both encoders generate powerful codes, but the one of the lower diagram enables the use of a simpler decoder.

$[b/(b+1)]$ binary convolutional code (or a short block code) with maximum free Hamming distance (or minimum distance). The interleaver (π) permutes the output of the outer encoder. The interleaved data enter the inner encoder, which implements a rate-(m/m) [rate-1] recursive convolutional code. The m output bits are then mapped to one symbol that belongs to a 2^m-level modulation. Because the inner code does not have redundancy, it is useless by itself; however, the combination of the inner and outer codes with the interleaver results in very powerful code. For MQAM where $M = N^2$, further reduction in complexity is possible. This can be done by assigning the $m = \log_2 N$ output bits of the inner encoder alternately to the in-phase and quadrature components of N^2QAM modulation. In this case, the bit-rate-to-bandwidth ratio will be

$$2bm/(b+1).$$

The advantage of this generic design can be made more apparent by citing an example of $b = 3$ for 16QAM, for which $m = 2$. In this example, the number of transitions per state of the inner TCM is only 4, which is only 1/32 of the corresponding number for the previous case.

This work was done by Dariush Divsalar, Sam Dolinar, and Fabrizio Pollara of Caltech for **NASA's Jet Propulsion Laboratory**. *Further information is contained in a TSP (see page 1)*

In accordance with Public Law 96-517, the contractor has elected to retain title to this invention. Inquiries concerning rights for its commercial use should be addressed to:

Intellectual Property group
JPL
Mail Stop 202-233
4800 Oak Grove Drive
Pasadena, CA 91109
(818) 354-2240

Refer to NPO-20878, volume and number of this NASA Tech Briefs *issue, and the page number.*

Enhanced Software for Scheduling Space-Shuttle Processing

Prototype software has been upgraded.

John F. Kennedy Space Center, Florida

The Ground Processing Scheduling System (GPSS) computer program is used to develop streamlined schedules for the inspection, repair, and refurbishment of space shuttles at Kennedy Space Center. A scheduling computer program is needed because space-shuttle processing is complex and it is frequently necessary to modify schedules to accommodate unanticipated events, unavailability of specialized personnel, unexpected delays, and the need to repair newly discovered defects. GPSS implements constraint-based scheduling algorithms and provides an interactive scheduling software environment. In response to inputs, GPSS can respond with schedules that are optimized in the sense that they contain minimal violations of constraints while supporting the most effective and efficient utilization of space-shuttle ground processing resources.

The present version of GPSS is a product of re-engineering of a prototype version. While the prototype version proved to be valuable and versatile as a scheduling software tool during the first five years, it was characterized by design and algorithmic deficiencies that affected schedule revisions, query capability, task movement, report capability, and overall interface complexity. In addition, the lack of documentation gave rise to difficulties in maintenance and limited both enhanceability and portability.

The goal of the GPSS re-engineering project was to upgrade the prototype into a flexible system that supports multiple-flow, multiple-site scheduling and that retains the strengths of the prototype while incorporating improvements in maintainability, enhanceability, and portability. The major enhancements were the following:

- The implementation of container objects (e.g., lists and maps) was made more efficient by use of the C++ Standard Template Library (STL).
- Improvements in the management of schedule network objects were made. An embedded schedule-data configuration-management subsystem, similar to systems used for software configuration management, was built. This subsystem accommodates multiple versions and revisions of each schedule, including direct descendants and branches. It also implements a concept of user sessions that enables each user to maintain multiple current instances of the same schedule and full schedule data files with sizes of the order of 1MB.
- Improvements in calendar operations were made. The original implementation required the full, time-series expansion of all calendars, giving rise to a large memory overhead. Furthermore, some calendar features (e.g., holidays), were "hard-coded." In the re-engineering, calendar memory requirements were reduced by providing for all calendar calculations to be performed in real time and by removing all hard-coded elements.
- Re-engineering of a robust query subsystem was perhaps the most challenging aspect of the project. The prototype utilized a Prolog-like query language that was scanned, parsed, and executed in a C program. The query code was problematic and difficult to understand. The re-engineering involved the building of a real (but similar) query language, utilizing the FLEX language and the Bison program to define a scanner and parser that includes all elements of logical inference (for example, AND, OR, and NOT) as well as full capability for building and incorporating user-customizable queries.
- An improved report architecture was developed. The prototype featured a significant number of hard-coded user options, and too little care was taken initially to develop a consistent but flexible report architecture. The re-engineering of the affected software components involved design around a new report class that contains attributes that describe the class of objects (e.g., tasks) represented in a report, the presentation style (e.g., Gantt chart or tabulation), and the time frame of the report. All report definitions are saved in files that the user can edit to customize reports.
- Several improvements in algorithms were made to solve backward-movement problems, provide a more robust implementation of achievers, and improve memory management through the use of smart pointers and "lazy load" of persistent data. Also included