航空技術學院學報 第十九卷　　第 27 －36 頁(民國 109 年)
Journal of Air Force Institute of Technology, Vol. 19 , pp. 27-36, 2020

使用交叉熵演算法建構接近最佳的可變長度錯誤更正碼

An Efficient Construction Strategy for Near-Optimal Variable-Length Error-Correcting Codes

吳峰蒼[1]、吳威龍[2]
Feng-Tsang Wu [1], Wei-Lung Wu [2]
[1] 空軍航空技術學院航空電子工程科 [2] 空軍航空技術學院管理系
[1] Department of Aero-Electronic Engineering, Air Force Institute of Technology.
[2] Department of Management, Air Force Institute of Technology.

摘要

在這篇文章，我們使用了高效率的交叉熵演算法去設計聯合訊號源編碼和通道編碼系統之可變長度的錯誤更正碼，這個演算法能夠讓我們建構出具有最短平均碼長及最低搜索複雜度的可變長度錯誤更正碼，這邊所持續強調的最短平均碼長會建構在各個不同的自由距離之下，交叉熵演算法可以按照不同的自由距離及不同編碼長度，設計出接近最佳的可變長度錯誤更正碼，最佳的可變長度錯誤更正碼代表是具有最短的平均碼長。

關鍵字：訊號源編碼和通道編碼系統、交叉熵演算法、可變長度的錯誤更正碼、最短平均碼長、自由距離。

Abstract

In this letter, we present an efficient cross-entropy (CE)-based algorithm for the design of variable-length errorcorrecting (VLEC) codes under the joint source and channel coding (JSCC) framework. The algorithm enables us to construct the near-optimal VLEC codes that have the minimum average codeword length (ACL) with low search complexity. The efficiency of the proposed CE-based algorithm makes it possible to construct the VLEC codes that have small ACL values under various freedistance constraints, especially for large-sized signal alphabets.

Keywords ： Joint source-channel coding (JSCC), variable-length error-correcting (VLEC) codes, cross-entropy method.

一、前言

現有的文獻已經證實「聯合訊號源編碼和通道編碼(Joint Source-Channel Coding， JSCC)」的系統性能及效率表現會比「分離訊號源編碼和通道編碼(Separate Source-Channel Coding，SSCC)」系統更好，特別是在系統有較嚴格之運算複雜度以及運算延遲的限制之下，JSCC有更好的優勢。現今數位通信系統採用訊號源編碼及通道編碼方案分開設計，是根據通信大師Shannon在1948年的信息理論中所建立之分離原則來規劃，說明訊號源編碼及通道編碼是可以分開設計，在傳輸資訊的過程中並不會影響系統的最佳化，所以一般在沒有任何損失的理想情況下，看成是個別獨立的部分來考量，系統將會達到最佳性且不會互相受到影響，但是，這個結果是被受質疑的。Zhong [1]證明將訊號源編碼及通道編碼是可以被整併設計，而且在許多實際情況下測試結果是超越原先的規劃，特別是在嚴格的延遲和硬體複

雜性限制下，均能表現更為出色。

這篇文章中我們展示了最佳的「整合來源和通道編碼」定義是能夠保證其錯誤更正能力，並且同時保持冗餘的能力，並盡可能保持較低的碼字長度。可變長度的錯誤更正碼(variable-length error-correcting codes，VLEC codes) [2]-[6]，具備了二項特性：

1. 具有將較短長度的碼字分配給高頻率出現的來源符號，因此降低了碼簿的平均碼字長度(average codeword length，ACL)。
2. 編碼的結構能夠進行錯誤更正，在[2]中顯示，自由距離(free distance，d_{free})是決定VLEC碼的錯誤更正能力的關鍵參數。

在先前的研究文章中[2]-[6]，以能設計出最短的ACL之VLEC碼為首要目標，換句話說，只要能搜尋出最小ACL值，就是最主要的貢獻。在本篇文章中，我們的目標是構建在聯合訊號源編碼和通道編碼系統之中，使用可變長度錯誤更正碼的技術，在固定自由距離的前提之下，設計演算法以搜尋具有最短的ACL之VLEC碼，設計出來之演算法可以大幅提高搜尋效率，降低搜尋時間。

在2013年，國立交通大學陳伯寧教授實驗室[3]提出了利用搜索樹(tree-search)演算法構建最佳化的VLEC碼並保證絕對可以找到文獻上所提出的最小ACL值。搜索樹演算法理論上是可以無限上綱的擴大搜索範圍且保證可以發現具有極高複雜度的最佳VLEC碼，但是實務上的執行是有困難的，因為在電腦模擬程式方面，當複雜度愈高時，所佔用的記憶體需求量必然會呈現指數性成長，然而實際電腦所使用的記憶體不可能是無限大，所以當複雜度持續升高時，必須執行演算法「優化搜索規則」的動作來降低對實際記憶體需求。因此，基於實體上的限制，陳教授們提出使用了樹狀路徑消除的概念，進一步設計了修改搜索樹演算法，然而，執行修改演算法的過程中，絕對有可能會刪除一些令人滿意的路徑，失去往最佳化結果靠近的機會。

另一方面，在[7]中提出了利用交叉熵(Cross Entropy，CE)演算法為通過一定隨機最佳化過程，找到與「真實分布相同」或「最接近」的預測分布，作為解決各種優化問題的方式，它是一種「機器深度學習法則」，可以非常迅速地得到最優化結果，求解的精準度也較其他方法來得更高，其原理是利用估計罕見事件的機率，來改善優化問題，經過實際研究證明[9]-[14]，已經帶來了許多令人滿意的解決方案。接下來，我們為了成功找尋大型來源字母表之接近最佳的VLEC碼，在本文中，我們使用了交叉熵演算方法的概念，並提出了能夠以26個英文字母及128位元的(American Standard Code for Information Interchange，ASCII)符號出現的機率來建構接近最佳VLEC碼的演算法。最後，我們建構提出的交叉熵演算法經實驗結果證明，與文獻中提供的結果相比，當考慮各個不同自由距離時，我們所提出的演算法能夠建構具有較小ACL值且接近最佳的VLEC碼。

二、 系統描述

現在假設有 M 個離散無記憶來源的字母符號 $A=\{\alpha_1,\alpha_2,...,\alpha_M\}$，其相對應的機率值直接給定為 $p_1,p_2,...,p_M$，$\sum_{i=1}^{M}p_i=1$，VLEC碼簿為二進制表示之 $C=\{\mathbf{c}_1,\mathbf{c}_2,...,\mathbf{c}_M\}$，並且 $\mathbf{c}_i$ 將對應到 α_i，接著我們第 i 個碼的長度定義成 l_i，最後定義ACL的計算公式為 $\omega=\sum_{i=1}^{M}p_i\cdot l_i$。

自由距離(free distance，d_{free})的定義，是可變長度的錯誤更正碼中最重要的核心參數，它是影響解碼錯誤率的主要原因。假設接收端知道接收到字元符號的數量是L與位元數的數量長度是N，$X_{L,N}$ 為L與N所形成的序列集合，其定義為：

$$X_{L,N}\equiv\left\{\underline{\mathbf{x}}=(\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_L):\forall\mathbf{x}_i\in C,\ \sum_{i=1}^{L}\left|l_{\mathbf{x}_i}\right|=N\right\}.$$

(1)

Buttigieg[2]，首先將自由距離定義為網格中相同節點上聚集的任意兩個不同序列$\underline{\mathbf{x}}$和$\underline{\mathbf{y}}$之間的最小漢明距離，自由距離的定義如下式：

$$d_{\text{free}}(C)=\min\left\{d_{\text{h}}(\underline{\mathbf{x}},\underline{\mathbf{y}}):\underline{\mathbf{x}},\underline{\mathbf{y}}\in X_N \text{ for some } N \text{ and } \underline{\mathbf{x}}\neq\underline{\mathbf{y}}\right\}. \quad (2)$$

其中$d_{\text{h}}(\underline{\mathbf{x}},\underline{\mathbf{y}})$表示具有相同長度的序列碼$\underline{\mathbf{x}}$和$\underline{\mathbf{y}}$之間的漢明距離。另外[3]在接收器處假設$L$和$N$的訊息是已知且可用的，將其定義修改為：

$$d_{\text{free}}(C)=\min\left\{d_{\text{h}}(\underline{\mathbf{x}},\underline{\mathbf{y}}):\underline{\mathbf{x}},\underline{\mathbf{y}}\in X_{L,N} \text{ for some } L,N \text{ and } \underline{\mathbf{x}}\neq\underline{\mathbf{y}}\right\}. \quad (3)$$

在[2]中也證明了d_{free}必需要滿足：

$$d_{\text{free}}(C)\geq\min\left\{d_{\text{b}}(C),d_{\text{div}}(C)+d_{\text{conv}}(C)\right\}. \quad (4)$$

在[3]中$d_{\text{b}}(C)$表示具有相同長度碼字之間的漢明距離：

$$d_{\text{b}}(C)=\min\left\{d_{\text{h}}(\mathbf{c}_i,\mathbf{c}_j):(\mathbf{c}_i,\mathbf{c}_j)\in C, \mathbf{c}_i\neq\mathbf{c}_j, \text{ and } l_i=l_j\right\}. \quad (5)$$

接著，$d_{\text{div}}(C)$是代表長度不相等的碼字，取碼字前段相等長度(靠左對齊)執行漢明距離計算，稱為最小發散距離(minimum diverging distance)，並且定義如下：

$$d_{\text{div}}(C)=\min\left\{d_{\text{h}}\left(c_i^1c_i^2...c_i^{l_j},c_i^1c_i^2...c_i^{l_j}\right):(\mathbf{c}_i,\mathbf{c}_j)\in C, \text{ and } l_i>l_j\right\}, \text{ with } \mathbf{c}_i=\left(c_i^1c_i^2...c_i^{l_i}\right). \quad (6)$$

最後，$d_{\text{conv}}(C)$是代表長度不相等的碼字，取碼字後段相等長度(靠右對齊)執行漢明距離計算，稱為最小收斂距離(minimum converging distance)，其定義如下：

$$d_{\text{conv}}(C)=\min\left\{d_{\text{h}}\left(c_i^{l_i-l_j+1}c_i^{l_i-l_j+2}...c_i^{l_i},c_i^1c_i^2...c_i^{l_j}\right):(\mathbf{c}_i,\mathbf{c}_j)\in C, \text{ and } l_i>l_j\right\} \quad (7)$$

在[2]中顯示，自由距離d_{free}是 VLEC 碼最重要的參數，它主導在高 SNR 區域中錯誤更正的能力，另外，作者也證明了，要產出令人滿意的 VLEC 碼，d_{div}和d_{conv}還需要去滿足以下的條件：

$$\begin{cases}\min\left\{d_{\text{b}}(C),d_{\text{div}}(C)+d_{\text{conv}}(C)\right\}\geq d^*_{\text{free}}, \text{ and}\\ \left|d_{\text{div}}(C)-d_{\text{conv}}(C)\right|\leq 1.\end{cases} \quad (8)$$

因此，我們這篇文章的目標是依數學式(8)建置接近最佳的VLEC碼，並專注於各種不同的自由距離約束下之最小的ACL值。換句話說，我們去建置各個不同自由距離的限制($d_{\text{free}}=3,5,7,9,11$)，可以依照數學式(8)找到目前文獻上最佳的VLEC碼。

三、 利用交叉熵演算法構建接近最佳的 VLEC 碼

交叉熵演算方法是一種蒙特卡洛(Monte Carlo method)方法，主要用來優化和重要性的取樣，在空間中按照某種規則執行撒點，將獲得每個點的誤差，再根據這些誤差信息決定下一輪撒點的規則，這過程稱為「迭代」，迭代是重複回饋過程的活動，其目的通常是為了接近到達所需的目標或結果。交叉熵方法是解決稀有事件估計和組合優化問題之有效方法，因為它具有有效的漸近收斂性及全局隨機搜索能力[7]。交叉熵方法主要的構想是使用以下兩個步驟將優化問題轉換為相關的隨機優化過程：

1. 以機率分佈函數產生一組候選者的樣本。
2. 依據現有的數據去更新分佈函數，以便在下一次迭代中產生更好的樣本。

在這個章節，我們討論利用交叉熵的方法來建構符合規則的VLEC碼，目標是之前所一直強調的「最小的ACL值」。我們所提出的方法是採用隨機特徵的方式，使得可以避免計算複雜度的升高，同時又有機會去搜索接近最佳的VLEC碼，交叉熵方法的構造方式在「演算法1」中有詳細的分析規劃，且關鍵步驟將在以下小節中介紹。

3.1 隨機樣本的產生

在 CE 的方法中使用伯努利分佈(Bernoulli distributions)去產生樣本向量「**s**」，**s** 向量中第 b 個元素，定義為$s(b)\in\{0,1\}$, $b=1,2,...,l_i$ ，伯

努利隨機變數 $P_{\mathrm{CE}}\left(s(b)=1\right)\equiv p_{\mathrm{CE}}(b)$ ，$0\le p_{\mathrm{CE}}(b)\le 1$ ，$1\times l_i$ 的矩陣可以被寫成 $\mathbf{p}_{\mathrm{CE}}=\left[p_{\mathrm{CE}}(1),p_{\mathrm{CE}}(2),...,p_{\mathrm{CE}}(l_i)\right]$，最初的機率向量矩陣給定為 $\mathbf{p}_{\mathrm{CE}}^{(1)}=0.5\cdot\mathbf{1}$，$\mathbf{1}$ 表示碼長長度為 l_i 的全為 1 之矩陣。$\mathbf{s}_r^{(t)}$ 表示有 2^{l_i} 個樣本 ($r=1,2,...,2^{l_i}$)，第 t 次迭代產生的機率向量 $\mathbf{p}_{\mathrm{CE}}^{(t)}$ 的是取決於 $(t-1)$ 次迭代後的影響。上述的表示式在演算法的步驟 4 及步驟 5 中呈現。

3.2 合格候選函數的計算

我們所提出的CE演算法在步驟6中，需要計算合格的候選者函數以評估和隔離不合格的候選碼字。假設已經找到(*i*-1)個合格的VLEC碼字，我們先定義 $\hat{C}\equiv\{\hat{\mathbf{c}}_1,\hat{\mathbf{c}}_2,\ldots,\hat{\mathbf{c}}_{i-1}\}$，$\ell(\cdot)$ 定義為計算候選向量函數中碼字的長度，碼字 $\hat{\mathbf{c}}_j\in\hat{C}$ ，$j=1,2,...,j-1$，

$$d\left(\mathbf{s},\hat{\mathbf{c}}_j\right)\equiv\begin{cases}d_{\mathrm{h}}\left(\mathbf{s},\hat{\mathbf{c}}_j\right), & \text{if } \ell(\mathbf{s})=\ell\left(\hat{\mathbf{c}}_j\right)\\ d_{\mathrm{d}}\left(\mathbf{s},\hat{\mathbf{c}}_j\right)+d_{\mathrm{c}}\left(\mathbf{s},\hat{\mathbf{c}}_j\right), & \text{if } \ell(\mathbf{s})\ne\ell\left(\hat{\mathbf{c}}_j\right)\end{cases} \tag{9}$$

因此，我們按上述的數學式(9)，當 $\ell(\mathbf{s})>\ell\left(\hat{\mathbf{c}}_j\right)$ 時，接著去定義

$$d_{\mathrm{d}}\left(\mathbf{s},\hat{\mathbf{c}}_j\right)\equiv d_{\mathrm{h}}\left\{\left[s(1),s(2),...,s(\ell(\hat{\mathbf{c}}_j))\right],\hat{\mathbf{c}}_j\right\}$$

$$d_{\mathrm{c}}\left(\mathbf{s},\hat{\mathbf{c}}_j\right)\equiv d_{\mathrm{h}}\begin{cases}\Big[s(\ell(\mathbf{s})-\ell(\hat{\mathbf{c}}_j)+1),s(\ell(\mathbf{s})-\ell(\hat{\mathbf{c}}_j)+2),\\ \qquad\qquad ...,s(\ell(\mathbf{s}))\Big],\hat{\mathbf{c}}_j\end{cases} \tag{10}$$

若要達成令人滿意的VLEC碼需要依照數學式(8)規則去完成每次候選碼的搜索，因此我們接著去表示，每個候選函數 $\mathbf{s}\in\Psi$ 且要滿足如下：

$$\begin{cases}\left[\min\limits_j d\left(\mathbf{s},\hat{\mathbf{c}}_j\right)\right]\ge d_{\mathrm{free}}^{*},\ \text{and}\\ \left|d_{\mathrm{d}}\left(\mathbf{s},\hat{\mathbf{c}}_j\right)-d_{\mathrm{c}}\left(\mathbf{s},\hat{\mathbf{c}}_j\right)\right|\le 1\end{cases} \tag{11}$$

3.3 迭代和終止條件的選擇

計算符合規則的候選函數用於每個碼字候選者 $\Psi\left(\mathbf{s}_r^{(t)}\right),r=1,2,...,2^{l_i}$ ，機率向量 $\mathbf{p}_{\mathrm{CE}}^{(t+1)}$ 的更新需滿足以下的方程式，

$$\mathbf{p}_{\mathrm{CE}}^{(t+1)}=\frac{\sum_{r=1}^{2^{l_i}}I\left\{\Psi\left(\mathbf{s}_r^{(t)}\right)\right\}\mathbf{s}_r^{(t)}}{\sum_{r=1}^{2^{l_i}}I\left\{\Psi\left(\mathbf{s}_r^{(t)}\right)\right\}}, \tag{12}$$

指標函數 $I\{\cdot\}$ 定義如下所列：

$$I\left\{\Psi\left(\mathbf{s}_r^{(t)}\right)\right\}=\begin{cases}1 & \text{if } \Psi\left(\mathbf{s}_r^{(t)}\right)\ge d_{\mathrm{free}}^{*}\\ 0 & \text{otherwise.}\end{cases} \tag{13}$$

因此，所有候選人的集合可分為兩組:『標準』和『不標準』，不標準的集合中，包含的樣本是不符合VLEC碼建構標準的碼字，因此無法成為VLEC碼。根據CE方法的原理，將更多合格者的樣本數放入『標準』集合中，是為了確保下一次迭代中產生的更好候選者，所以『不標準』集合中的樣本將被忽略，並且將不參與機率向量的更新。

在持續不斷的迭代過程中，機率向量 $\mathbf{p}_{\mathrm{CE}}^{(t)}$ 將會漸漸逐步的收斂，持續到最後，如果可以收集到可靠且滿意的集合，則 $\mathbf{p}_{\mathrm{CE}}^{(t)}$ 中的每個位置元素，預期會收斂到 1 或 0，因此該位置成為全「1」的向量，就判斷為二進制的 1，反之則為 0，就可以產生出一組二進制的碼字。

接下來，我們要來說明「演算法1」中，「步驟8」需要驗證三種情況：

(一) 假設『標準』的集合中的碼字成為「空矩陣」，這種情況是非常糟糕的，這意味著已經沒有任何一個合格的候選碼字在隨機生成的樣本中，這表示著根據給定的碼字長度 l_i 搜索範圍，已經找不到任何可能的解決方案，所以現在我們需要去增加 l_i 的長度，但不只是單純只有探討 l_i 的長度，而是連排序在 l_i 之後的長度 l_k 也需要去討論，例如：當增加完 l_i 之後，卻發現 $l_i>l_k$ ($k>i$ 是可以被滿足的)，然而這是違反我們一開始所制訂的規則

$\left(l_1 \le l_2 \le \cdots \le l_M\right)$ ，所以至少要設定長度 $l_i = l_k$，接下來，就再次返回到步驟3，重新執行搜索第 i 個碼字的過程，以完成符合標準VLEC碼。

(二) 假設『標準』的集合中的碼字不是「空矩陣」，但是某個位置的元素，持續一直無法收斂到「0」或「1」，在這種情況下，我們將重新執行另一次迭代，以根據先前的規則，重新生成一組新的隨機樣本 $\mathbf{p}_{\mathrm{CE}}^{(t)}$ 。

(三) 最後，假設如果 $\mathbf{p}_{\mathrm{CE}}$ 中所有元素的值已經穩定了，則 $\hat{\mathbf{c}}_i = \mathbf{p}_{\mathrm{CE}}^{(t)}$ 將是確定為第 i 個合格的VLEC碼字。

演算法 1：

利用 CE 演算法建構接近最佳的 VLEC 碼

步驟 1：設定要搜索的字母數 M 、U_b 是目前文獻上的最短 ACL 及需要的 d_{free}^{*}，接著我們隨機產生 M 個碼長 $l_1 \le l_2 \le \ldots \le l_M$ 。

$$\begin{cases} d_{\mathrm{free}}^{*} \le l_1 \le \left(d_{\mathrm{free}}^{*}+1\right) \\ \omega = \left(\sum_{i=1}^{M} p_i \cdot l_i\right) \le U_b \end{cases} \tag{14}$$

步驟 2：設定變數 i=1 及產生第 1 個二進制的碼字且長度為 l_1 。

步驟 3：計算平均碼長 ω 。

If $\omega > U_b$，返回步驟 1.

Else $i = i+1$.

步驟 4：迭代變數 t=1 及最初的 $\mathbf{p}_{\mathrm{CE}}^{(1)} = 0.5 \cdot \mathbf{1}$。

步驟 5：使用機率質量函數 $g\left(\mathbf{s};\mathbf{p}_{\mathrm{CE}}^{(t)}\right)$ 去隨機產生 2^{l_i} 個二進制的樣本 $\mathbf{s}_r^{(t)}, r = 1,2,...,2^{l_i}$.

步驟 6：合格候選函數的計算

For $r = 1$ **to** 2^{l_i}

For $j = 1$ **to** $i-1$

根據方程式(9)及(10)計算 $d\left(\mathbf{s}_r^{(t)}, \hat{\mathbf{c}}_j\right)$ 取代 $\mathbf{s}_r^{(t)}$ 和 $\hat{c}_j$

End For j

依據(15)去得到 $\Psi\left(\mathbf{s}_r^{(t)}\right)$

End For r

步驟 7：依據(12)和(13)更新機率向量 $\mathbf{p}_{\mathrm{CE}}^{(t+1)}$ 。

步驟 8：

If 『滿意』的集合是空集合，

設定 $l_i = \left(l_i + 1\right)$

For $z = 1,2,...,\left(M-i\right)$

If $\left(l_{i+z} < l_i\right)$ 更新 $\left(l_{i+z} = l_i\right)$

End For z

設定 $i = i-1$ 並且返回步驟 3

Else if 『滿意』的集合不是空集合及在 $\mathbf{p}_{\mathrm{CE}}^{(t)}$ 內的任一元素還沒收斂到『0』或『1』

設定 $t = t+1$ 並且返回步驟 5

Else $\hat{c}_i = \mathbf{p}_{\mathrm{CE}}^{(t)}$

步驟 9：

If $\left(i = M \text{ and } \omega \le U_b\right)$，演算法結束

Else 返回步驟 3

四、模擬結果

首先，我們要來解釋在本文中所一直闡述的「接近最佳」，這專業術語的用意。陳伯寧教授等人[3]提出了利用搜索樹演算法構建「最佳化」的VLEC碼，他們利用8個符號組成的小字母表，來展示該演算法能夠測試建構VLEC碼的所有可能結果，以此證明他們的方法是「最佳」，所找到VLEC碼的最小ACL值為7.24。然而，當論文[2]和[4]都用提出的演算法去建構8個符號的VLEC碼，所找到的最小ACL值分別為7.864及7.936，接著，我們用交叉熵演算法可以建構的ACL值為7.304，僅比最佳值多0.064位元，因此，我們就聲稱可以使用所提出的演算法去建構「接近最佳」的VLEC碼。

接下來，我們考慮兩種信號源來呈現比較的結果，分別是26個符號的英文字母和128個符號的ASCII字母。我們在表格 1中，對於26個符號英文字母表的比較，我們依[3]所建構的方式，給出了針對不同的d_{free}去構建二種不同機率分布的VLEC碼ACL值，當與文獻[2]-[5]中提供的結果相比較時，通過使用我們所提出的交叉熵演算法可以獲得比較小的(或更好的)ACL值。接著在考慮128符號ASCII字母時，我們比較文獻[2]，[4]和[5]是使用貪婪搜索演算法，是極不可能用來搜索位元數高的的VLEC碼，儘管可以獲得可接受的結果，例如[2]中提供的結果，但是隨著字母表大小的增加，需要費時相當多的搜索時間來獲得接近最優的解決方案，如表格 2所示。對於英語字母表的呈現，在表格 3及表格 4中，我們展示了二種不同的機率分布針對不同的$d_{\text{free}}=5, d_{\text{free}}=7, d_{\text{free}}=9, d_{\text{free}}=11$值去構建VLEC碼的位元組合呈現。

在[3]中驗證了與[2]，[4]和[5]文獻中的方法相比，搜索樹演算法實現了更少的搜索時間，因此，在表2中，我們比較了交叉熵和搜索樹演算法的搜索時間和電腦使用的記憶體大小，對於字母表較小的來源，搜索樹算法所需的搜索時間是小於交叉熵演算方法的搜索時間，然而，隨著字母表個數的增加，這種情況會迅速增長，此外，搜索樹算法需要非常大的電腦記憶體空間來建構具有可比較的VLEC碼ACL值。由於當前的硬體限制，我們無法使用搜索樹演算法來構建128個符號的比較結果，相反的，我們所提出的交叉熵演算法僅需要較低的電腦記憶體來儲存使用，演算法實現了隨機搜索方式，使得可以在合理的搜索時間內，搜索到大尺寸字母表的VLEC碼。因此，當將我們的結果與[2]中提供的兩個設計結果進行比較時，可以觀察到顯著的差別，如表格 2所示。

五、 結論

在這篇文章中，提出了一種有效的設計演算法『交叉熵，CE』演算法來為了建置可靠傳輸的近似最佳VLEC碼，實驗數值結果呈現，CE演算法，絕對可以有效率的搭配隨機搜索特性，搜索到大字母表的VLEC碼並且與文獻中的設計方法相比，我們的CE演算法可以獲得優異或可比較的結果。

表格 1. 26 個符號英文字母表和 128 個符號 ASCII 表的 VLEC 碼平均碼長(ACL)

演算法	[2]	[4]	[5]	[3]	交叉熵	[2]	交叉熵
符號個數	26	26	26	26	26	128	128
機率1	ACL	ACL	ACL	ACL	ACL	ACL	ACL
$d_{\text{free}}=3$	6.4038	6.4047	6.3574	6.2560	6.2529	N/A	7.9513
$d_{\text{free}}=5$	8.4740	8.5049	8.4740	8.3223	8.2609	9.7286	9.6946
$d_{\text{free}}=7$	10.5388	10.5110	10.5388	10.3615	10.3406	15.1107	11.9997

$d_{\text{free}}=9$	12.8898	12.9644	12.8898	12.6647	12.5554	N/A	15.1559
$d_{\text{free}}=11$	15.0345	15.0846	15.0345	14.6521	14.6509	N/A	16.8706
機率2	ACL	ACL	ACL	ACL	ACL		
$d_{\text{free}}=3$	6.272617	6.309980	6.266612	6.189350	6.188550		
$d_{\text{free}}=5$	8.378035	8.400986	8.378035	8.333866	8.279556		
$d_{\text{free}}=7$	10.559646	10.599945	10.488923	10.302508	10.26768		
$d_{\text{free}}=9$	12.737255	12.806644	12.737255	12.532291	12.47091		
$d_{\text{free}}=11$	14.876166	15.354549	15.024952	14.580329	14.54564		

表格 2. $d_{\text{free}}=7$ 的 VLEC 碼建構效率比較表

演算法	符號個數	8	26	32	64	128
[3]	記憶體	4GB	8 GB	8 GB	32 GB	N/A
	時間	4s	33m20s	40m26s	5d9h37m42s	N/A
	ACL	7.24	10.3615	10.092	11.5309	N/A
交叉熵	記憶體	4GB	8 GB	8 GB	8 GB	8 GB
	時間	20m	20m	2m	12m	33m
	ACL	7.304	< 10.3615	< 10	< 11.8	< 12

表格 3. 26 個符號英語字母表機率分布 1 的 VLEC 碼

符號	機率	$d_{\text{free}}=5$	$d_{\text{free}}=7$	$d_{\text{free}}=9$	$d_{\text{free}}=11$
$\alpha_1=\text{E}$	0.1270	001010	00111101	0000010111	00111001000
$\alpha_2=\text{T}$	0.0906	110101	11000010	1111101000	000100101111
$\alpha_3=\text{A}$	0.0817	0011011	000100001	00001011001	0101110111100
$\alpha_4=\text{O}$	0.0751	1100100	111011110	11110100110	1010011000011
$\alpha_5=\text{I}$	0.0697	00000101	0100110000	000110000101	00100100110001
$\alpha_6=\text{N}$	0.0674	11111010	1010000111	110001101010	11001011101110
$\alpha_7=\text{S}$	0.0633	010000111	00000111010	0011001110010	000001110010011
$\alpha_8=\text{H}$	0.0609	100011100	10101000001	0110110001100	111010001100100
$\alpha_9=\text{R}$	0.0599	011111001	01011011101	00111100101101	0100111001001011
$\alpha_{10}=\text{D}$	0.0425	101100010	11110100110	01011111010010	1110000010110101
$\alpha_{11}=\text{L}$	0.0403	0001000000	001010101000	10000000000000	10001111110011001
$\alpha_{12}=\text{C}$	0.0278	0100111010	011101100111	11100011111111	01000000001010100
$\alpha_{13}=\text{U}$	0.0276	1001101111	110110010100	001010101100100	11110111011100010
$\alpha_{14}=\text{M}$	0.0241	1110001100	100001010011	011101110111011	011011100111010111
$\alpha_{15}=\text{W}$	0.0236	1010010001	0010011000101	100110011000011	100000010110100100
$\alpha_{16}=\text{F}$	0.0223	0111000111	0111001111011	100101000011100	100111101000001010
$\alpha_{17}=\text{G}$	0.0202	00011100011	1000110011011	0110000010111000	010001011001111001
$\alpha_{18}=\text{Y}$	0.0197	01100100000	1101010110100	0001111010001101	1000110010110110000
$\alpha_{19}=\text{P}$	0.0193	10000010100	1011101001000	1011010100100011	1100000101001110111
$\alpha_{20}=\text{B}$	0.0149	11101111011	01011110110001	1100101111010011	0110111111011101001
$\alpha_{21}=\text{V}$	0.0098	01011011000	01100001001101	0010001001000110	1111001010000011010
$\alpha_{22}=\text{K}$	0.0077	10111101101	10110110101111	1101110101111100	01101010101001100000
$\alpha_{23}=\text{J}$	0.0015	100000110110	01001001110110	01101001100011101	10001101011111110100
$\alpha_{24}=\text{X}$	0.0014	101111000000	010001011111010	10010010111100011	01100011011010011111
$\alpha_{25}=\text{Q}$	0.0010	0100111101001	100110111101000	10101111010001010	11010100000000010010
$\alpha_{26}=\text{Z}$	0.0007	1110111110000	011110100001110	11010001011010100	100010100100010100110

表格 4. 26 個符號英語字母表機率分布 2 的 VLEC 碼

符號	機率	$d_{\text{free}}=5$	$d_{\text{free}}=7$	$d_{\text{free}}=9$	$d_{\text{free}}=11$
$\alpha_1=\text{E}$	0.148786	001111	00110101	0000001100	00001110110
$\alpha_2=\text{T}$	0.093541	110000	11001010	1111110011	011010000000
$\alpha_3=\text{A}$	0.088337	1011000	000011001	00101010111	0001101100101
$\alpha_4=\text{O}$	0.072458	0100111	111100110	11010101000	1110010111010
$\alpha_5=\text{I}$	0.068722	00000101	0010101110	000111000010	00110101000111
$\alpha_6=\text{N}$	0.064985	11111010	1101010001	011001111101	11000010011001
$\alpha_7=\text{S}$	0.058313	011100110	00010010000	0001100111000	010101001011000
$\alpha_8=\text{H}$	0.056445	100101101	01100000111	0111011000110	111111110100111
$\alpha_9=\text{R}$	0.055378	111011011	10101111010	00110101101111	1000010110101001
$\alpha_{10}=\text{D}$	0.043768	0101010011	11011101101	01001000010011	0101001101010000
$\alpha_{11}=\text{L}$	0.041233	1001101101	010001111100	10001110100100	00110001111001010
$\alpha_{12}=\text{C}$	0.027622	0010001010	101111101001	11110011011000	11001111001100011
$\alpha_{13}=\text{U}$	0.025754	0110100100	010110010001	010110010111101	10110010010110101
$\alpha_{14}=\text{M}$	0.024553	1111011110	101000000100	101000001000001	010110011111011111
$\alpha_{15}=\text{W}$	0.023619	00010011101	0110011000001	111011110010110	011101100110100100
$\alpha_{16}=\text{F}$	0.020817	11011101011	1000000110010	000101101100110	101000111000110001
$\alpha_{17}=\text{G}$	0.018682	10100111110	0001101101100	0101101010010110	1001001111110011101
$\alpha_{18}=\text{Y}$	0.015212	00001000000	01101001010010	0010110101001010	1010011000111000010
$\alpha_{19}=\text{P}$	0.015212	01100100001	01010110101011	1100000111111101	1101110010101111000
$\alpha_{20}=\text{B}$	0.012677	101001100011	10100110111100	10000101111001111	10010000100011001001
$\alpha_{21}=\text{V}$	0.011609	011010001100	10000001011111	10011010010010011	01100011101110101100
$\alpha_{22}=\text{K}$	0.008674	110111000001	10111000101010	10100100001110000	10111100001100111010
$\alpha_{23}=\text{J}$	0.001468	101010111111	011111100011100	10111011100101110	10101001011111010101
$\alpha_{24}=\text{X}$	0.000801	0101100111111	100001110010010	100001000111011001	011111011110100010101
$\alpha_{25}=\text{Q}$	0.000801	1010101100010	011110001000011	011110111000100111	100000001011111001110
$\alpha_{26}=\text{Z}$	0.000534	0001001001100	100100001011110	100011111100111010	00100011011001111011

參考文獻

[1] Y. Zhong, F. Alajaji, and L. L. Campbell, “On the joint source-channel coding error exponent for discrete memoryless systems,” IEEE Trans. Inf. Theory, vol. 52, no. 4, pp. 1450-1468, Apr. 2006.

[2] V. Buttigieg, “Variable-length error-correcting codes,” Ph.D. thesis, Univ. of Manchester, England, 1995.

[3] T.-Y. Wu, P.-N. Chen, F. Alajaji, and Y. S. Han, “On the design of variable-length error-correcting codes,” IEEE Trans. Commun., vol. 61, no. 9, pp. 3553-3565, Aug. 2013.

[4] C. Lamy and F. X. Bergot, “Lower bounds on the existence of binary error-correcting variable-length codes,” in Proc. 2003 IEEE Inform. Theory Workshop, pp. 300-303.

[5] J. Wang, L.-L. Yang, and L. Hanzo, “Iterative construction of reversible variable-length codes and variable-length error-correcting codes,” IEEE Commun. Lett., vol. 8, no. 11, pp. 671-673, Nov. 2004.

[6] H. Hijazi, A. Diallo, M. Kieffer, L. Liberti, and C. Weidmann, “A MILP approach for designing robust variable-length codes based on exact free distance computation,” in Proc. 2012 Data Compression Conference, 10-12 Apr. 2012, Snowbird, USA, pp. 257-266.

[7] R. Y. Rubinstein, “Optimization of computer simulation models with rare events,” European Journal of Operations Research, 99:89-112, 1997.

[8] R. Y. Rubinstein and D. P. Kroese, “The cross-entropy method.” Berlin, Germany: Springer, 2004.

[9] K. Chepuri, T. Homem-de-Mello, “Solving the vehicle routing problem with stochastic demands using the cross-entropy method,” Annals of Operations Research, vol. 134, No. 1, pp. 153-181, 2005.

[10] J.-C. Chen, M.-H. Chiu, Y.-S. Yang, and C.-P. Li, “A suboptimal tone reservation algorithm based on cross-entropy method for PAPR reduction in OFDM systems,” IEEE Trans. Broadcast., vol. 57, no. 3, pp. 752–756, Sep. 2011.

[11] J.-C. Chen, S.-H. Wang, M.-K. Lee, and C.-P. Li, “Cross-entropy method for the optimization of optical alignment signals with diffractive effects,” IEEE J. Lightw. Technol., vol. 29, no. 18, pp. 2706-2714, Sep. 15, 2011.

[12] J.-C. Chen, C.-K. Wen, C.-P. Li, and P. Ting, “Cross-entropy optimization for the design of fiber Bragg gratings,” Photonics Journal, IEEE, vol. 4, no. 5, pp. 1495 –1503, oct. 2012.

[13] K.-C. Lee, S.-H. Wang, C.-P. Li, H.-H. Chang, and H.-J. Li, “Adaptive resource allocation algorithm based on cross-entropy method for OFDMA systems,” IEEE Transactions on Broadcasting, vol. 60, no. 3, pp. 524–531, Sept 2014.

[14] W. J. Huang, W. W. Hu, C. P. Li, and J. C. Chen, “Novel metric-based PAPR reduction schemes for MC-CDMA systems,” IEEE Trans. Veh. Technol., vol. 64, no. 9, pp. 3982–3989, Sep. 2015.

[15] Y.-M. Huang, T.-Y. Wu, and Y. S. Han, “An A*-based algorithm for constructing reversible variable-length codes with minimum average codeword length,” IEEE Trans. Commun., vol. 58, no. 11, pp. 3175-3185, Nov. 2010.

[16] A. Diallo, C. Weidmann, and M. Kieffer, “Optimizing the free distance of error-correcting variable-length codes,” in Proc. 2010 IEEE Int. Workshop Multimedia Signal Process., pp. 245–250.