计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.2014.20121213
51(8):1739-1747, 2014

基于像素链动态失真和校验格码的±1 隐写编码

包震坤[1,2] 张卫明[1,3] 程 森[1,2] 赵险峰[4,5]
[1](信息工程大学四院 郑州 450002)
[2](数学工程与先进计算国家重点实验室(信息工程大学) 郑州 450002)
[3](中国科学技术大学信息科学技术学院 合肥 230026)
[4](中国科学院信息工程所 北京 100093)
[5](信息安全国家重点实验室(中国科学院信息工程研究所) 北京 100093)
(bao252547116@sina.com)

±1 Steganographic Codes by Applying Syndrome-Trellis Codes to Dynamic Distortion Model in Pixel Chain

Bao Zhenkun[1,2], Zhang Weiming[1,3], Cheng Sen[1,2], and Zhao Xianfeng[4,5]
[1](*The Fourth Institute, Information Engineering University, Zhengzhou* 450002)
[2](*State Key Laboratory of Mathematical Engineering and Advanced Computing (Information Engineering University), Zhengzhou* 450002)
[3](*School of Information Science and Technology, University of Science and Technology of China, Hefei* 230026)
[4](*Institute of Information Engineering, Chinese Academy of Sciences, Beijing* 100093)
[5](*State Key Laboratory of Information Security (Instituteof Information Engineering, Chinese Academy of Sciences), Beijing* 100093)

Abstract Double-layered STC (syndrome trellis code) is the most popular method for minimizing the distortion of ±1 steganography. However, it is a probabilistic algorithm which may fail in the embedding process on some given profiles. Another characteristic of double-layered STC is the high computational complexity. Starting from these two points, we propose a dynamic distortion model defined in a pixel chain in this paper. The dynamic distortion model is working on a principle that the SLSB (second least significant bit) of current pixel is used to control the LSB (least significant bit) of the next pixel. So the distortion of some pixels may be adjusted to zero by this means. We apply STC to fit the dynamic distortion model and get a novel method for ±1 steganography. Comparing with the double-layered STC, the experiment result shows that the proposed method has comparable ability for minimizing distortion with significantly improved embedding speed. And this novel method avoids failure in the embedding process. Considering the advantages together, the method is more suitable for steganography systems and software in practical environment.

Key words information hiding; steganographic codes; double-layered embedding; pixel chain; syndrome trellis codes

摘 要 双层校验格码(syndrome trellis code, STC)是目前最小化±1 隐写失真的流行工具,但是该方法是概率算法,可能造成嵌入失败,并且计算复杂度较高.因此提出像素链动态失真模型,利用当前像素的 SLSB(second least significant bit)位控制下一像素的 LSB(least significant bit)位,使得嵌入过程中有些像素的失真可以动态调整为零.进而,将 STC 码应用于动态失真模型,得到了一种新的±1 隐写编码

收稿日期:2012-12-25;**修回日期**:2013-08-27
基金项目:国家自然科学基金项目(60803155,61170234);中国科学院战略性先导科技专项课题(XDA06030601)

方法.新方法最小化失真的能力与双层 STC 接近,嵌入速度明显优于双层 STC.并且新方法是确定性算法,可以保证嵌入成功,所以更适合在实际的应用系统中使用.

关键词 信息隐藏;隐写编码;双层嵌入;像素链;校验格码

中图法分类号 TP309

隐写术是信息隐藏技术的一个重要分支,其目的是在公开的多媒体数据(如数字图片)中嵌入秘密消息从而进行隐蔽通信.发送者通过对载体图片中的元素(如像素或量化 DCT 系数)等进行微小的修改来嵌入秘密消息.隐写术的安全强度是指抵抗隐写分析检测的能力,最小化嵌入失真是提高隐写安全性的主要手段之一.

要最小化嵌入失真,首先根据载体特征设计一个失真函数,对每个载体元素定义嵌入失真(反映该元素被修改后造成的影响程度),从而消息嵌入过程转变成一个优化,其问题的目的是对于给定的消息长度,最小化嵌入失真.例如,对于空域图像,可用纹理复杂度定义失真函数,对于同样的修改幅度,纹理越复杂的区域造成的失真值越小.

最小化失真隐写的一个自然方法是将消息嵌入到失真最小的区域(比如纹理最复杂区域).但是,因为消息嵌入过程会对像素做修改,因而改变失真函数的值,这样接收者就无法同步失真信息来定位消息嵌入位置,所以无法提取消息.虽然可以通过密钥选取初始空间来确定最终的嵌入位置来解决[1],但并不能从根本上消除.Fridrich 等人提出通过编码技术解决,比如湿纸编码[2](wet paper codes, WPC)或校验格码[3](syndrome-tellis codes, STC).利用湿纸编码,接收者将失真小的元素定义为"干点",失真大的原素定义为"湿点",然后通过只修改干点嵌入消息,而接收者无需知道"干点"的位置,即可提取消息.显然,湿纸编码是针对两级失真(干/湿)函数设计的,而 STC 码则是适用于各种失真函数.对于定义好的失真和给定的消息长度,STC 码可以最小化嵌入失真,并使得接收者在无需同步失真信息的条件下提取消息.因而,STC 码是目前设计自适应隐写的最便利的工具之一.

由于隐蔽性的要求,隐写术对载体元素的修改幅度必须很小.因而±1 是最流行的隐写修改方式,它通过+1 或−1 的方式修改载体,修改幅度仅为 1.通过±1,可以修改像素的最低有效位(least significant bit, LSB)来表示消息,这种隐写称为 LSB Matching.后来,研究者发现,通过±1,可以同时用像素的 LSB 位和 SLSB(second least significant bit)位表示消息.因为通过选择+1 或−1 可以修改 LSB 位并控制 SLSB 位的取值,所以能够同时在 SLSB 位嵌入消息,而不需引入额外修改.但是由于接收者无法知道哪些像素需要±1 修改,所以 SLSB 的嵌入需要使用湿纸编码,这一方法被称为双层嵌入(double layered embedding, DLE)[4].

双层嵌入也可以用 STC 码实现,在第 2 层,将湿点的失真定义为无穷大即可.但是,当 STC 编码用于湿纸模型时,若湿点比率过高或嵌入率过大,嵌入失败的概率也会很大.为此,Filler 对双层嵌入作了推广[5],推广后的方法将嵌入步骤颠倒,先在 SLSB 层嵌入,然后在 LSB 层嵌入,只有第 2 步需要引入湿点,湿点的比率被大大降低,所以 STC 编码嵌入失败概率可以显著降低.

近年来,许多图像自适应隐写方法采用了 STC 码提高安全性.著名的空域图像隐写算法 HUGO[6]使用二元 STC 码嵌入消息.最近 Holub 等人[7]提出的 WOW 算法基于方向滤波定义失真,并分别尝试二元 STC 码和双层 STC 码最小化嵌入失真.文献[7]中的实验结果表明基于双层 STC 的方案,其安全性明显优于基于二元 STC 的方案,并且也明显优于 HUGO 算法.文献[8-9]中提出的 JPEG 图像的自适应隐写算法也都采用了双层 STC 码.

综上,使用双层 STC 进行±1 嵌入是提高隐写安全性的有效手段.但是使用双层 STC 时需要计算每层的嵌入率并对各层重新计算失真,这使得实现过程较为复杂.并且双层 STC 存在嵌入失败的情况,这也增加了实际应用中的复杂性.

为解决上述问题,本文提出了一种新的基于 STC 的±1 隐写编码方法,该方法只执行一次二元 STC 码即可完成嵌入.为此,我们首先扩展 LMR 方法[10],将相邻像素的 SLSB 位与 LSB 位异或,通过这种方式把所有像素链接在一起,从而像素的两层 LSB 被"拉伸"成一条二元载体,我们称这条新载体为"像素链",然后在像素链上执二元 STC 编码.

利用 STC 编码序贯嵌入的特点,我们在编码过程中动态调整像素链上的失真定义来减少嵌入失真.

事实上,由于±1可以修改像素的LSB位并同时控制SLSB位的取值,所以在像素链上,当两个相邻比特都需要修改时,只需要修改一个像素就可以完成对这两个比特的修改.因而,如果当前像素需要修改,则下一个像素的失真可以调整为0.通过失真的动态调整,可以引导STC编码搜索到失真更小的嵌入路径.该方法的特点是充分发挥±1修改的优势但不需要引入湿点,因而保证STC编码过程总能成功.所以本文方法在实际应用中,实现更为简便,不需要处理嵌入失败的情况.并且,该方法在保持最小化失真能力与双层STC基本相当条件下,显著提高了嵌入速度.

1 基础知识

本文以灰度图像为例描述信息嵌入方法,但本文方法同样适用于其他载体,比如JPEG图像的量化DCT系数.

1.1 双层嵌入

双层嵌入(DLE)[4],是一种利用像素LSB位和SLSB位分别嵌入消息的隐写方法.该方法的具体思想是:修改某个像素灰度值 g 的LSB位,可以通过 $g+1$ 或者 $g-1$ 来完成.在这两种修改方式之间进行选择即可控制 g 的SLSB位的取值,从而用SLSB位表示1 b消息.例如,设 $g=4$(二进制表示为$_2(100)$),当其LSB位需要修改为1时,可以将 g 修改为3(二进制表示为$_2(011)$)或者5(二进制表示为$_2(101)$).如果需要在其SLSB位嵌入1,则选择修改为3;如果需要在其SLSB位嵌入0,则选择修改为5.

DLE的实现过程如下:首先在载体LSB层进行二元嵌入,假设嵌入 l 个比特消息后有 N 个像素需要修改,此时并不立即对这 N 个像素进行修改,而是记录这 N 个像素的位置;在SLSB层,标注的 N 个位置记为干点,其他位置记为湿点,然后使用湿纸编码,大约可嵌入 N 个比特消息.湿纸编码过程会确定这 N 个像素的SLSB位是否需要修改,从而确定对每个标注像素采用+1还是−1修改方式,同时完成两层的嵌入.上述方法在不增大修改幅度和不增加修改量的前提下,在像素的SLSB层多嵌入大约 N 个比特消息.换言之,对于给定的消息长度,DLE方法可以有效减少修改个数.

1.2 基于STC的最小化失真隐写

以二元载体为例,设对于一个 n 长二元载体 $\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$,存在一个失真函数 ρ,对于每个载体元素 x_i,$1\leqslant i\leqslant n$,都定义了一个度量嵌入失真的值 $\rho(x_i)=\rho_i\geqslant 0$,其值越大意味着修改造成的影响越大.嵌入失真$\{\rho_i\}_{1\leqslant i\leqslant n}$是依据某些边信息来定义的,比如像素周围的纹理复杂度.

如果我们在 $\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$中嵌入 l 个比特消息 $\boldsymbol{m}=(m_1,m_2,\cdots,m_l)$,得到了载密对象 $\boldsymbol{y}=(y_1,y_2,\cdots,y_n)$,则嵌入率定义为 $\alpha=l/n$,平均失真定义 μ_c 为:

$$\mu_c=\frac{1}{n}\sum_{i=1}^{n}\rho_i\delta(x_i,y_i), \tag{1}$$

其中,

$$\delta(x_i,y_i)=\begin{cases}1, & \text{if } x_i\neq y_i;\\ 0, & \text{if } x_i=y_i.\end{cases} \tag{2}$$

编码的目的是对给定的嵌入率 α,最小化平均失真 μ_c,或等价为最大化嵌入效率 α/μ_c.

Filler等人在文献[3]中提出的STC码,对于给定的嵌入失真$\{\rho_i\}_{1\leqslant i\leqslant n}$和嵌入率,可以接近嵌入效率上界.STC码是一种特殊的矩阵编码,所用的矩阵 $\boldsymbol{H}$ 由一个 $h\times w$ 子矩阵 $\hat{\boldsymbol{H}}$ 以一种级联的方式拼接而成.其目的是将 $\boldsymbol{x}$ 修改成 $\boldsymbol{y}$,使得 $\boldsymbol{H}\boldsymbol{y}^{\mathrm{T}}=\boldsymbol{m}^{\mathrm{T}}$,从而接收者通过计算 $\boldsymbol{H}\boldsymbol{y}^{\mathrm{T}}$ 即可提取消息.STC编码将 $\boldsymbol{H}\boldsymbol{y}^{\mathrm{T}}=\boldsymbol{m}^{\mathrm{T}}$ 的每个解都表示成通过栅格的一条路径,路径每一步的长度由嵌入失真$\{\rho_i\}_{1\leqslant i\leqslant n}$定义,从而对式(1)的最小化就变成了寻找最短路径问题,而最短路径可以由维特比译码的方式快速得到.

如在第1节中指出的,使用STC编码执行双层嵌入时,第2层湿率太大会导致编码失败,Filler为了解决这个问题,提出一种巧妙的解决方案[5]:首先,使用熵的链式法则将嵌入率分解,估计出SLSB层需要承载的嵌入率,然后算出SLSB层在该嵌入率下每一比特的最优修改概率;然后,利用用于嵌入信息的翻转引理并调用STC编码嵌入消息;接着,在SLSB层被修改的条件下,计算LSB层每一个比特的条件修改概率,在此过程中,SLSB位已修改的像素所对应的LSB位应被定义为湿点(即不允许修改的点);最后,再次利用翻转引理并调用STC编码嵌入消息.

这样实现的双层嵌入因为使用STC编码在SLSB层嵌入消息后,其修改率很低,所以在LSB层引入的湿点比率也很低,从而使STC编码的失败概率大幅度降低.但嵌入时的湿度仍然大于零,所以双层STC编码依旧是一个概率算法.如果发生嵌入

失败的情况,Filler 选择将嵌入消息减少 1 b,并初始化随机种子置乱,然后再进行一次嵌入,虽然这样做最终能保证消息嵌入,但是在实际使用中会带来问题. 比如在一副图像中嵌入一个完整的文件,如果嵌入失败,不能简单地将文件数据流减少 1 b,因为这可能造成提取的文件无法打开. 所以使用双层 STC 进行实际应用时需要匹配的的协议来处理嵌入失败的异常. 另外,因为双层 STC 需要计算出每一个像素 LSB 位和 SLSB 位的修改概率和失真函数,这导致速度较低.

下面我们基于 STC 提出一种确定性的 ± 1 隐写编码,既能避免嵌入失败的情况,又能明显提高嵌入速度.

2 像素链动态失真算法

本节,我们首先引入一个像素链动态失真模型,然后改造 STC 编码以适应该模型,得到动态失真编码算法.

2.1 像素链动态失真模型

假设一个由 n 个像素组成的灰度图像载体 $\boldsymbol{g}=(g_1,g_2,\cdots,g_n)$,对应的像素 g_i 的修改失真定义为 ρ_i,$1\leqslant i\leqslant n$. 我们用 $L(g_i)$ 表示取 g_i 的 LSB 位,用 $SL(g_i)$ 表示取 g_i 的 SLSB 位. 生成像素链的具体过程为:像素 g_i 的 LSB 位 $L(g_i)$ 与像素 g_{i-1} 的 SLSB 位 $SL(g_{i-1})$ 异或构成二元载体的第 i 个比特 x_i 的值,$i=2,3,\cdots,n$,载体第 1 个比特等于 g_1 的 LSB 位 $L(g_1)$,即:

$$x_1=L(g_1),$$
$$x_i=SL(g_{i-1})\oplus L(g_i),\ i=2,3,\cdots,n. \quad (3)$$

图 1 给出了一个包含 8 个像素的例子,像素值为:$\boldsymbol{g}=(g_1,g_2,\cdots,g_8)=(3,5,7,4,2,3,7,6)$,由像素链构成的二元载体为 $\boldsymbol{x}=(x_1,x_2,\cdots,x_8)=(1,0,1,1,0,0,0,1)$.

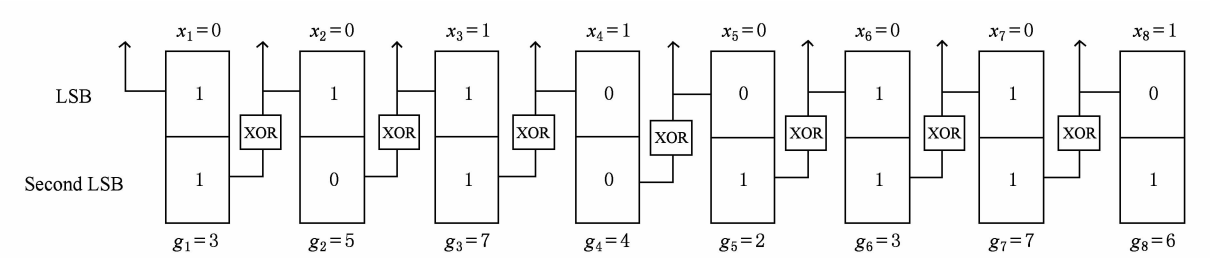


Fig. 1 The procedure of generating binary.

图 1 二元载体生成过程示意图

接下来,我们在二元载体 $\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$ 上嵌消息,首先,将像素 g_i 的失真 ρ_i 赋给 x_i,然后在嵌入过程中,根据像素的修改情况,来动态调整失真 ρ_i 的取值. 其原理是:若 x_1 需要修改,只能通过 g_1+1 或 g_1-1 完成,修改方向由 g_1 的取值和 x_2 是否需要修改决定:

1) 当 g_1 是奇数,若 x_2 需要修改,则取 g_1+1;若 x_2 不需要修改,则取 g_1-1.

2) 当 g_1 是偶数,若 x_2 需要修改,则取 g_1-1;若 x_2 不需要修改,则取 g_1+1.

而对于 x_i,$2\leqslant i\leqslant n$,若 x_i 需要修改,首先看 g_{i-1} 是否需要修改,如果 g_{i-1} 需要修改,则通过 ± 1 修改 g_{i-1} 的 SLSB 位可以完成对 x_i 的修改. 如果 g_{i-1} 不需要修改,则必须通过 $g_i\pm 1$ 完成对 x_i 的修改,修改方向由 g_i 的值和 x_{i+1} 是否需要修改决定:

1) 当 g_i 是奇数,若 x_{i+1} 需要修改,则取 g_i+1;若 x_{i+1} 不需要修改,则取 g_i-1.

2) 当 g_i 是偶数,若 x_{i+1} 需要修改,则取 g_i-1;若 x_{i+1} 不需要修改,则取 g_i+1.

例如,我们要在图 1 所示的载体 $\boldsymbol{x}=(x_1,x_2,\cdots,x_8)=(1,0,1,1,0,0,0,1)$ 上通过替换嵌入 8 b 消息 $\boldsymbol{m}=(1,1,0,0,0,1,1,1)$,也就是说需要将 $\boldsymbol{x}$ 修改成 $\boldsymbol{m}$. 因为 $x_1=m_1$,所以 x_1 不需要修改,故 g_1 也不需修改;x_2 需要修改,必须通过修改 g_2 完成,此时需要观察 x_3,因为 x_3 需要修改且 g_2 是奇数,所以将 g_2 改为 $g_2+1=6$;g_4 需要修改,而 x_5 不需修改,g_4 为偶数,所以将 g_4+1 变为 5;x_6 需要修改,因 x_7 需要修改,g_6 是奇数,所以将 g_6+1 变为 4;x_8 不需修改,所以 g_8 不变. 最后修改后的像素序列 $(g_1,g_2,\cdots,g_8)$ 变为 $(3,6,7,5,2,2,7,6)$,从而二元载体序列 $\boldsymbol{x}$ 也修改成了需要嵌入的消息 $\boldsymbol{m}$. 上述过程嵌入消息只修改 3 个像素,而如果直接在像素的 LSB 嵌入消息,则需要修改 5 个像素.

像素链可以减少修改个数的原因是:当像素 g_i 需要修改时,可以自由控制 x_{i+1} 的取值,即使 x_{i+1} 需要改动,也不需修改 g_{i+1},从而减少 1 个修改. 换言

之,此时修改 x_{i+1} 是没有失真代价的,所以对应的嵌入失真 ρ_{i+1} 可以重新赋值为 0.

对于饱和的像素(如 $g=0$ 或 255)需要特殊处理.我们采取双层嵌入中常用的方法:当 $g=0$,而 g 需要减 1 时,取 $g+3$;当 $g=255$,而 g 需要加 1 时,取 $g-3$.这样也可以同时完成对 LSB 位和 SLSB 位的修改,但是修改幅度变大.一般而言,上述两种情况出现的概率较小,对隐写算法整体性能的影响可以忽略.

2.2 采用动态失真的 STC 编码

对于给定的二元载体和消息,以及定义好的载体失真,STC 编码可以用失真尽量小的修改方式嵌入消息.STC 编码逐比特决定每个载体比特是否需要修改,所以我们可以先利用像素链生成二元载体,然后在二元载体上执行 STC 编码.编码过程中,动态调整失真定义,即如果当前像素 g_i 需要修改,则像素链下一比特载体 x_{i+1} 对应的失真 ρ_{i+1} 可以重新赋值为 0.为此我们还需要对 STC 码的编码过程作适当调整,使其沿着更小的失真路径进行.

在 STC 编码中,校验矩阵 $\boldsymbol{H}$(嵌入矩阵)是由规模为 $h\times w$ 的子矩阵 $\hat{\boldsymbol{H}}$ 生成的,生成方式如图 2 中所示,将数个子矩阵 $\hat{\boldsymbol{H}}$ 从左至右依次降行排列生成校验矩阵.子矩阵的高 h 和宽 w 是由使用者自己选择的,高 h 影响嵌入效率和嵌入速度,h 越高嵌入效率越高但是嵌入速度越低,而宽 w 决定嵌入率 $\alpha=1/w$.

我们用图 2 中的例子来说明 STC 算法和动态失真 STC 算法的不同之处.图 2(a)为 STC 编码过程,图 2(b)为基于动态失真的 STC 编码过程,两个例子中采用同样的载体序列、消息序列、嵌入矩阵和载体失真定义.为了描述简便,载体失真全部定义为 1.但是需要注意图 2(a)中的载体是一般的二元序列(比如像素的 LSB 位),而图 2(b)中的载体是由像素序列 $\boldsymbol{g}=(3,5,7,4,2,3,7,6)$ 通过像素链生成的.

我们将载密对象 $\boldsymbol{y}=(y_1,y_2,\cdots,y_n)$ 视为变量,记校验子为 $\boldsymbol{s}=(s_1,s_2,\cdots,s_l)$ 且满足 $\boldsymbol{H}\boldsymbol{y}^{\mathrm{T}}=\boldsymbol{s}^{\mathrm{T}}$.嵌入 l 长消息 $\boldsymbol{m}=(m_1,m_2,\cdots,m_l)$ 的过程就是找到一个相对于载体 $\boldsymbol{x}=(x_1,x_2,\cdots,x_n)$ 修改失真最小的载密对象 $\boldsymbol{y}$,使得校验子满足 $\boldsymbol{s}=\boldsymbol{m}$.

从嵌入矩阵 $\boldsymbol{H}$ 的构造我们可以发现,因为 $\boldsymbol{H}$ 第 1 行只有前 $w(w=2)$ 位为 1,其他位全为 0,所以 $\boldsymbol{y}$ 的前 w 位就能确定 $\boldsymbol{s}$ 的第 1 位.同样,$\boldsymbol{y}$ 的前 $2w$ 位能确定 $\boldsymbol{s}$ 的第 2 位,并且能类似地推广到所有 $\boldsymbol{s}$ 中的位.另一方面,因为 $\boldsymbol{H}$ 每一列非零元个数都不超过 $h(h=2)$,所以 $\boldsymbol{y}$ 中任一位 y_i 在 $\boldsymbol{s}$ 中所能影响的校验子位数也不超过 h.具体来说,y_i 影响 (s_2,s_1),y_2 影响 (s_3,s_2).所以每个变元 y_i 影响 h 个校验子的状态,所有可能的值有 2^h 个,在图(2)中用 *State* 表示.

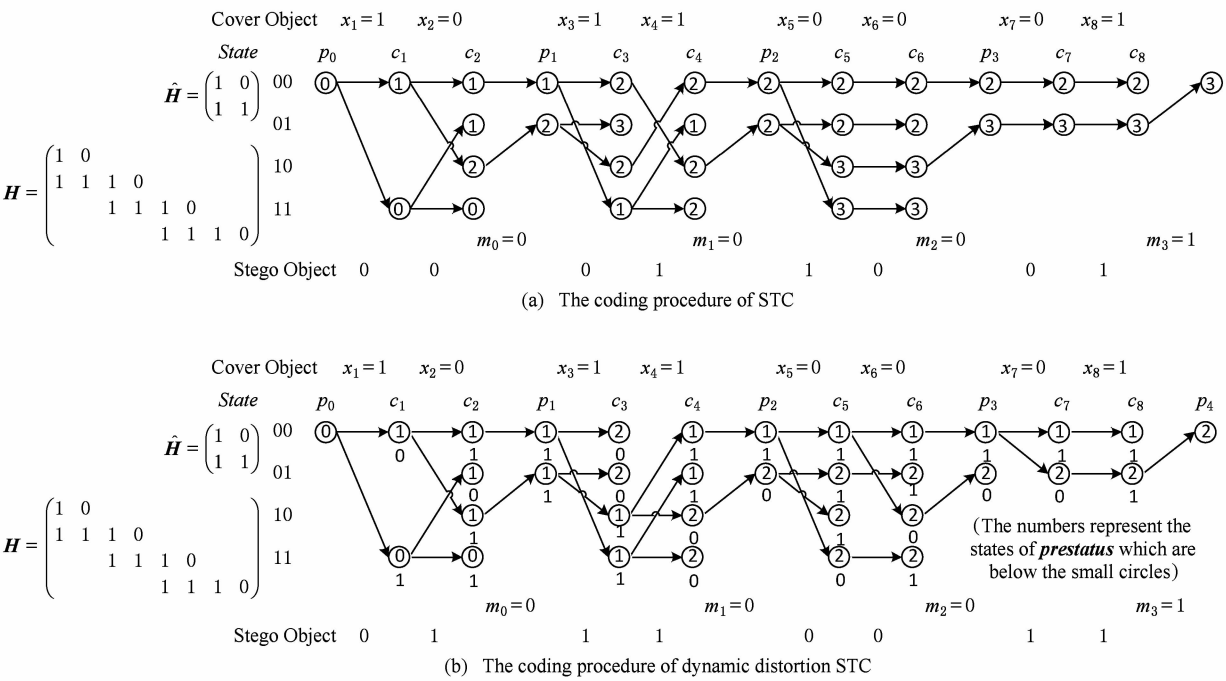


Fig. 2 Example of STC and dynamic distortion STC.

图 2 STC 与动态失真 STC 的例子

为了动态调整失真,我们附加一个对应校验子 2^h 个可能状态的向量 $\boldsymbol{prestatus}=(r_0,r_1,\cdots,r_{2^h-1})$,其中每个分量 $r_i\in\{0,1\}$,$0\leqslant i\leqslant 2^h-1$ 表示状态 i 所在的嵌入路径上前一像素是否需要修改.按照2.1节描述的像素链动态失真机制,如果前一个像素需要修改,那么当前载体比特即使修改也不会引入失真,所以可以将其失真重新定义为0,那么该嵌入路径对应 $\boldsymbol{prestatus}$ 的分量 r_j,$j\in\{1,2,\cdots,n\}$ 变为0,反之为1.这时,STC会遍历所有状态来寻找失真最小的路径,所以当有些载体比特失真归0后,STC会找到失真更小的路径.

STC编码中的一个问题是:如果两条路径到达某一状态的总失真相同,应该选择哪条路径?原始STC的解决方案是选择任何一条都可以.但是对于基于动态失真的STC,两条路径对后面的嵌入过程造成的影响可能不一样.对于这种情况,我们选择路径的准则是:选择到达该状态时需要修改当前像素的那条路径.因为如果当前像素需要修改,那么下一个载体比特的失真可以赋值为0,因而可能生成失真更小的路径.

图2所示的两个例子中,载体为(1,0,1,1,0,0,0,1),消息为(0,0,0,1).因为消息长度为4 b,所以编码过程能够由4个小块组成的格子表示,每个小块由4行3列共 $2^h(w+1)=4\times3=12$ 个节点组成,每一块对应载体的2个比特.第1个块中的节点代表校验子 $\boldsymbol{s}$ 的前2个比特(s_2,s_1)在嵌入过程中的状态,嵌入路径走过第1块以后,校验子的第1比特将被确定,即 $s_1=m_1$.进入第2块中的节点状态对应的比特更新为(s_3,s_2),后面各块的状态依次类推.例子中两种嵌入方法,都用(00,01,10,11)这4个状态($State$)来指示 $\boldsymbol{s}$ 中2个比特的变化.在动态失真STC中,需要另外定义一个四元向量 $\boldsymbol{prestatus}$ 来表示像素是否需要修改,向量 $\boldsymbol{prestatus}$ 中各个分量的脚标分别用其对应的状态(00,01,10,11)表示,$\boldsymbol{prestatus}$ 各分量初始值均定义为1(表示像素不需修改),即 $\boldsymbol{prestatus}(00,01,10,11)=(1,1,1,1)$.

图(2)中每块的第1列代表进入该块时对应校验子比特的初始状态,并依次将4小块中的第1列标记为 p_0,p_1,p_2 和 p_3.4块中其他8列节点,对应校验矩阵 $\boldsymbol{H}$ 的8列,依次标记为$\{c_1,c_2,\cdots,c_8\}$,其中第 i 列上的节点表示:添加 $\boldsymbol{H}$ 的第 i 列(即取 $y_i=1$),或不添加 $\boldsymbol{H}$ 的第 i 列(即取 $y_i=0$)对校验子 $\boldsymbol{Hy}$ 状态的影响.

格子中的每一条路径表示为一个满足 $\boldsymbol{Hy}^{\mathrm{T}}=\boldsymbol{s}^{\mathrm{T}}=\boldsymbol{m}^{\mathrm{T}}$ 前数个等式的载密 $\boldsymbol{y}$,每条路径从最左边全零状态 $p_0(s_2s_1)=p_0(00)$(p_0 列,00状态)开始,是否加 $\boldsymbol{H}$ 的第1列将引出2条从 $p_0(00)$连到 c_1 列2个状态的边,如果不加上 $\boldsymbol{H}$ 的第1列(即令 y_1 为0),则(s_2s_1)的状态保持00,因此我们画一条连接 $p_0(00)$和 $c_1(00)$的线段,因为原载体 $\boldsymbol{x}$ 第1个比特是1,所以在这个情况下需要修改载体第1个比特,因此在两个算法中这条边均赋予代价"1".因为对应像素需要修改,所以在动态失真算法中还要将 $\boldsymbol{prestatus}(00)$由1变为0.如果加上 $\boldsymbol{H}$ 的第1列(即令 y_1 为1),则 s_2s_1 的状态从00变成11,因此我们画一条连接 $p_0(00)$和 $c_1(11)$的线段,在这个情况下不需要修改载体第1个比特,因此这条边赋予重量"0"并且在动态STC算法中 $\boldsymbol{prestatus}(11)$的值保持1.

接着,从 c_1 列的 $c_1(00)$和 $c_1(11)$开始,通过选择添加或不添加 $\boldsymbol{H}$ 的第2列,可以分别到达 c_2 列中对应的两个节点.例如从 $c_1(00)$出发,不添加 $\boldsymbol{H}$ 的第2列,到达 $c_2(00)$添加 $\boldsymbol{H}$ 的第2列,到达 $c_2(10)$.注意,在动态STC算法中,因为 $\boldsymbol{prestatus}(00)=0$,所以由 $c_1(00)$引出的2条边重量都为0,即此时不管是否添加 $\boldsymbol{H}$ 的第2列,都不需要修改第2个像素,所以对应地,要令 $\boldsymbol{prestatus}(00)=1$,$\boldsymbol{prestatus}(10)=1$.所有到达 c_2 的路径添加完成后,STC算法到达 $c_2(00)$的路径总代价为1,达到 $c_2(01)$的路径总代价为1,达到 $c_2(10)$的路径总代价为2,达到 $c_2(11)$的路径总代价为0;而使用动态失真STC算法,达到 $c_2(00)$的路径总代价为1,达到 $c_2(01)$的路径总代价为1,达到 $c_2(10)$的路径总代价为1,达到 $c_2(11)$的路径总代价为0.

因为校验子 $\boldsymbol{s}$ 第1个比特 s_1 只被 $\boldsymbol{H}$ 中开始的两列影响,所以在完成第1块之后,每条路径对应的 s_1 的值已经确定,需要满足 $s_1=m_1$,所以在本例中,我们要满足 $s_1=m_1=0$,即要将到达 $c_2(01)$和 $c_2(11)$两条路径舍弃掉,此步骤称为"剪枝".一般地,在每个小块的最后,都有一半路径被剪掉,剩下的一半路径进入下一块.本例第2个块表示(s_3s_2)的状态,第2块的 p_1 列中会有两个初始状态连接上一块保留下来的两条路径.在动态失真算法中,也要将 $\boldsymbol{prestatus}$ 中对应被保留下来状态的值依次保存在 $\boldsymbol{prestatus}(00)$和 $\boldsymbol{prestatus}(01)$中.在本例中,就是在令 $\boldsymbol{prestatus}(00)=\boldsymbol{prestatus}(00)$,$\boldsymbol{prestatus}(01)=\boldsymbol{prestatus}(10)$之后,将 $\boldsymbol{prestatus}(10)$和 $\boldsymbol{prestatus}(11)$置为1.

用类似方法可生成由 c_2 到 c_3 的路径以及后续的路径.对于到达同一状态的两条路径,只保留总代价小的那条.如果两条路径代价相同,对于 STC 可以保留任意一条;但是对于动态失真 STC,保留哪一条要由 ***prestatus*** 决定.例如在图 2(b)中,到达 $c_4(10)$ 的两条路径一条是来自 $c_3(00)$,另一条是来自 $c_3(10)$,并且到达 $c_4(10)$ 后路径总代价都为 2,我们选择来自 $c_4(10)$ 的路径,原因如下:在第 c_3 列中,***prestatus***(00)=0(即第 3 个像素需要修改),所以由 $c_3(00)$ 引出的路径没有失真代价(即第 4 个像素不需修改);而由 $c_3(10)$ 到 $c_4(10)$ 的路径将导致第 4 个像素被修改,从而由 $c_4(10)$ 引出的路径将没有失真代价,即第 5 个载体比特的失真可以调整为 0.

第 3 块中的路径生成同第 2 块类似.而在最后一块中,因为 $\boldsymbol{H}$ 的第 7 和第 8 列只能影响 s_4,所以第 4 小块中的节点只能代表 s_4 的两个状态,这保证最后只有一个路径到达最右边的状态.因为 $\boldsymbol{H}$ 的第 8 列是全零的,而且 $x_8=1$,所以两个编码过程中我们均令 $y_8=x_8=1$.

完成上述过程后就是进行回溯来构造载密序列.可以看出图 2(a)(b)到达 p_4 列的都只有一条路径,从该路径往 p_0 列回溯从而构造出唯一的载密序列 $\boldsymbol{y}$.STC 和动态失真 STC 构造出的载密 $\boldsymbol{y}$ 分别为:(0,0,0,1,1,0,0,1)和(0,1,1,1,0,0,1,1).和载体 $\boldsymbol{x}$=(1,0,1,1,0,0,0,1)相比,STC 构造出的载密修改了 3 个位置,动态失真 STC 也是修改了 3 个位置.但是注意到动态失真 STC 的载体 $\boldsymbol{x}$ 是由 $\boldsymbol{g}$=(3,5,7,4,2,3,7,6)通过像素链生成的.按照上节提出的像素链修改规则,需对像素作如下修改:$y_1\neq x_1$ 且 $y_2\neq x_2$,$g_1=3$ 是奇数,所以 g_1 修改为 4 的同时 g_2 不变,而 y_3,y_4,y_5,y_6 与 x_3,x_4,x_5,x_6 均相同,所以 g_3,g_4,g_5,g_6 不变,而 $y_7\neq x_7$ 但是 $y_8=x_8$,所以 g_7 修改为 6 的同时 g_8 不变,这样得到一条载密 ***sc***=(4,5,7,4,2,3,6,6),与原载体相比只修改了 2 个位置.

因为本文的方法从根本上来说仍然是一种二元编码,所以本文方法支持的嵌入率范围是 $0<\alpha\leqslant 1$.

STC 嵌入可以利用维特比算法快速实现,包含前置和回溯两个部分.前置部分通过格子生成嵌入路径,回溯部分构造出满足条件的最优载密.其中的细节在文献[3]中描述,并且从 http://dde.binghamton.edu/download/syndrome/能得到 STC 的源代码.

类似地,动态失真 STC 也可以利用维特比算法快速实现,我们在图 3 中给出了动态失真 STC 算法的伪码.

```
/* Forward Part of the Viterbi Algorithm */
wght[0]=0;
wght[1,…,2^h-1]=INFINITY;
prestatus[0,1,…,2^h-1]=1;
indx=indm=1;
for i=1,…,num of blocks (submatrices in H) {
  for j=1,…,w { /* for each column */
    for k=0,…,2^h-1 { /* for each state */
      w0=wght[k]+x[indx]×rho[indx]×prestatus[k];
      w1=wght[k XOR H_hat[j]]+(1-x[indx])×
          rho[indx]×prestatus[k XOR H_hat[j]];
      path[indx][k]=w1<w0? 1:0;
      prestatus[k]=(1-prestatus[k])+prestatus[k] ×
          (1-(path[indx][k] XOR x[indx]));
      newwght[k]=min(w0, w1);
    }indx++, wght=newwght;} }
/* Prune States */
for j=0,…,2^(h-1)-1 {
  prestatus[j]=precgstatus[2×j+message[indm]];
  prestatus[2^(h-1),…,2^h-1]=INFINITY;
  wght[j]=wght[2 × j+message[indm]];
  wght[2^(h-1),…,2^h-1]=INFINITY;
  indm++;}
/* Backward Part of the Viterbi Algorithm */
embedding_cost=wght[0];
state=0, indx--, indm--;
for i=num of blocks,…,1{ /* step-1 */
  for j=w,…,1{ /* step-1 */
    y[indx]=path[indx][state];
    state=state XOR (y[indx]×H_hat[j]);
    indx--;
  }
  state=2×state+message[indm];
  indm--;
}
/* LEGEND
INPUT: x, message, H_hat
x=(x[1],…,x[n]) is Cover Object;
message=(message[1],…,message[m]);
H_hat[j]=j th column in Int Notation;
OUTPUT: y, embedding_cost
y=(y[1],…,y[n]) is Stego Object; */
```

Fig. 3 Pseudocode of the Viterbi algorithm modified for applying STC to dynamic distortion model.
图 3 动态失真 STC 中维特比算法伪码

3 实验结果

我们在 Filler 的 STC 源代码基础之上对本文提出的算法在 C++上进行了实现,下面的实验结果

在配置CPU为Intel Core™ i5 2.67GHz、内存4GB的个人电脑上得到的.

Filler提供了2种版本的STC源码,分别针对单字节整型(Char型)失真和单精度浮点型(Float型)失真,整型失真可以实现更快速的嵌入.由于双层STC需要将修改概率转换成失真并定义无穷失真标记湿点,所以Filler后来提出的双层STC只能调用针对浮点型失真的STC程序.而本文方法不需引入湿点,并且不存在失真转换的问题,所以同时适用于整型失真和浮点型失真.

我们分别对失真值为整型和浮点型的两种情况实现了两个版本的动态失真STC,首先,我们将实现的这两个版本与STC的两个版本以及Filler提出的双层STC在相同条件下对嵌入速度进行了比较.表1比较了5种编码在相同的矩阵高度($h=6,7,8,9,10$)和相同的嵌入率($\alpha=0.5$)下的嵌入速度,即每秒能完成嵌入的载体长度(单位为Mbps).可以看出,STC的速度最快,本文方法采用浮点型失真时的速度明显优于双层STC,并且在整型失真下可以获得更快的嵌入速度.而嵌入速度对于某些隐写应用,比如基于即时通信的隐蔽通信系统[11-12],是非常重要的.

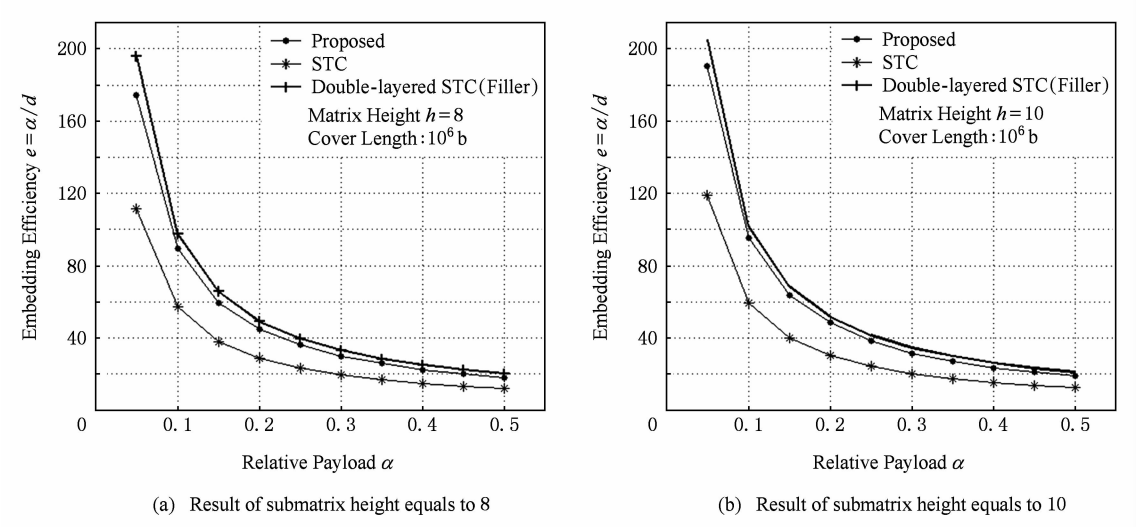
Table 1 Comparison of Five Types STC at Throughput

表1 五种类型STC算法嵌入速度比较结果 Mbps

Code Type	Submatrix Height h				
	6	7	8	9	10
STC(Float)	6.6622	3.8226	2.1749	1.0959	0.6046
STC(Char)	12.8205	8.0120	4.9261	3.0488	1.6447
Double-layer STC	0.6105	0.5574	0.4613	0.3392	0.2218
Proposed (Float)	4.5872	2.5641	1.3089	0.6748	0.3446
Proposed (Char)	10.7527	6.4103	4.2735	2.4691	1.3089

然后我们进行了嵌入效率的实验,失真值通过随机抽样得到,满足区间[0,1]上的均匀分布.对3种编码取子矩阵高度$h=8$和10,在不同嵌入率($\alpha=0.05\sim0.5$)下分别进行嵌入效率的实验.实验过程中,每种嵌入率下测试1000条长度为10^5的随机载体,每次嵌入的消息序列也是随机生成的,然后对嵌入效率取平均结果.

如图4所示,双层STC和本文方法都可以获得明显高于STC的嵌入效率,而本文方法与双层STC的嵌入效率非常接近.综合表1可知,对比于双层STC,本文方法在保持嵌入效率的前提下,可以明显提升嵌入速度.另外,本文方法不需引入湿点,不会造成嵌入失败的情况,所以可以降低系统的复杂性,更便于在实际的隐写系统中应用.



(a) Result of submatrix height equals to 8 (b) Result of submatrix height equals to 10

Fig. 4 Comparison of STC, double-layer STC and proposed method at embedding efficiency.

图4 本文方法与STC,double-layer STC在嵌入效率上的比较结果

4 总 结

本文提出了一种像素链动态失真模型,并基于此模型设计一个适于±1 嵌入的 STC 编码方法. 与 STC 和双层 STC 比较,本文方法在嵌入效率和嵌入速度之间取得了很好的折中. 该方法的嵌入效率与双层 STC 接近,但嵌入速度有明显提升. 另外,本文方法是一个确定性算法,弥补了双层 STC 存在嵌入失败概率的不足,使得本文方法更适合在实际应用系统中使用.

参 考 文 献

[1] Gui Xinlu, Li Xiaolong, Yang Bin. A content-adaptive ±1-based steganography by minimizing the distortion of first order statistics [C] //Proc of 2012 IEEE ICASSP. Piscataway, NJ: IEEE, 2012: 1781-1784

[2] Fridrich J, Goljan M, Lisonek P, et al. Writing on wet paper [J]. IEEE Trans on Signal Processing, 2005, 53(10): 3923-3935

[3] Filler T, Judas J, Fridrich J. Minimizing embedding impact in steganography using trellis-coded quantization [C] //Proc of IS&T/SPIE Electronic Imaging, Media Forensics and Security XII. Bellingham: SPIE, 2010: 1-14

[4] Zhang Xinpeng, Zhang Weiming, Wang Shen. Efficient double-layered steganographic embedding [J]. Electronics Letters, 2007, 43(8): 482-483

[5] Filler T, Judas J, Fridrich J. Minimizing additive distortion in steganography using syndrome-trellis codes [J]. IEEE Trans on Information Forensics and Security, 2011, 6(4): 920-935

[6] Pevny T, Filler T, Bas P. Using high-dimensional image models to perform highly undetectable steganography [C] // Proc of the 12th Int Workshop Information Hiding. Berlin: Springer, 2010: 161-177

[7] Holub V, Fridrich J. Desiging steganographic distortion using directional filters [C] //Proc of 2012 IEEE Int Workshop on Information Forensic and Security. Piscataway, NJ: IEEE, 2012: 234-239

[8] Filler T, Fridrich J. Design of adaptive steganographic schemes for digital images [C] //Proc of SPIE, Electronic Imaging, Media Watermarking, Security, and Forensics XIII. Bellingham: SPIE, 2011: 1-14

[9] Cheng Sen, Zhang Weiming, Bao Zhenkun, et al. Based the magnitude of DCT coefficient adaptive steganographic method for JPEG images [J]. Journal of Wuhan University: Natural Science Edition, 2012, 58(6): 545-550 (in Chinese)
(程森, 张卫明, 包震坤, 等. 基于 DCT 系数大小关系的自适应 JPEG 隐写[J]. 武汉大学学报: 理学版, 2012, 58(6): 545-550)

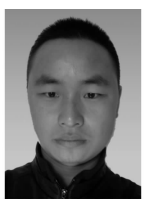
[10] Mielikainen J. LSB matching revisited [J]. IEEE Signal Processing Letters, 2006, 13(5): 285-287

[11] Yuan Jian, Zhou Xuesi, Huang Yongfeng. Application of HOOK technology in network covert communication [J]. Journal of Sichuan University: Natural Science Edition, 2011, 48(3): 539-545 (in Chinese)
(袁键, 周学思, 黄永锋. HOOK 技术在网络隐蔽通讯中的应用[J]. 四川大学学报: 自然科学版, 2011, 48(3): 539-545)

[12] Shirali-Shahreza M H, Shirali-Shahreza S. Real-time and MPEG-1 layer III compression resistant steganography in speech [J]. IET Information Security, 2010, 4(1): 1-7

Bao Zhenkun, born in 1989. Master candicate in the Information Engineering University. Student member of China Computer Federation. His main research interest is information hiding.

Zhang Weiming, born in 1976. Associate professor and master supervisor in the University of Science and Technology of China. His main research interests include information hiding and network security.



Cheng Sen, born in 1988. Received his master degree in the Information Engineering University. His main research interest is information hiding.

Zhao Xianfeng, born in 1969. Professor in the Institute of Information Engineering, Chinese Academy of Sciences. His main research interests include information hiding and network security.