Collberg の Software Watermarking 法の実装と評価

指導教官：市川周一　　学籍番号：023707 小竹徹

1　はじめに

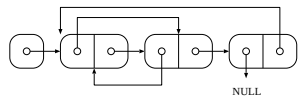
電子透かしとは，ソフトウェアの中に著作権などの情報を密かに埋め込む手法である．Collberg と Thomborson [1] は，プログラム実行時に動的領域にグラフを構築して情報を埋め込む方法を提案した．彼らは静的透かしより動的透かしのほうが秘匿性が高いと主張している．しかし，グラフを構成する情報を付加するため，プログラムサイズが大きくなる．

本研究の目的は，Collberg の考案した 2 つの手法とそれを応用した手法を実装し，プログラムサイズの増加と透かし埋め込み時間を定量的に評価することである．

2　手法

Radix−k 法 [1] は，透かしの k 進法表現をリスト構造で表す．始点を除いた $(k-1)$ ノードで，0〜$(k^{k-1}-1)$ までの数を表現する．リストを構成するノードは，次ノードをさすポインタと，係数を表すポインタをもつ．係数ポインタは，0 を表したいとき NULL，1 を表したいとき自分自身を指す．$k \geq 2$ 以降を表すときは $(k-1)$ 先のノードを指す．図 1 に，$k=4$ の例を示す．

二つ目は数え上げ法 [1] である．有向木のパターン数は，構成ノード数に応じて Knuth の式 [2] によって数え上げることができる．透かし値 n を超えるパターンをもつ最小のノード数を求め，Collberg の番号付け規則 “largest subtree first” [1] に基づいて，有向木に番号を割り当てる (図 2)．n 番目のグラフを構成することで，透かし値 n を表現する．



$$44 = 2\cdot4^2 + 3\cdot4^1 + 0\cdot4^0$$

図 1: Radix−k 法 ($k=4$)

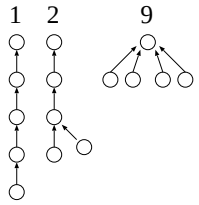


図 2: 数え上げ法 (5 ノード)

加算数え上げ法は，数え上げ法の実装を簡略化するために著者が考案した手法である．Knuth の手法で有向木を数え上げる点は数え上げ法と同様である．ただし，埋め込みには単純な形の有向木 (根から出ている部分木のノード数が変化する木) だけを用いる．数え上げ法と同じ方法で数を割り当てると，割り当て可能な数は飛び飛び (離散的) になる．そこで複数の有向木を用い，それらの合計で透かし値を表現する．

3　実験

本研究で作成したプログラムは，C 言語のソースファイルを読み込み，透かしを構成するコードおよび認識するコードを付加して，変更された C 言語ソースを出力する．埋め込み可能な数は，Radix−k 法では制限なし，数え上げ法と加算数え上げ法では 10 進 9 桁 (30 bit) までである．

例題プログラム (`whetstone.c`) に対して，桁数を 1〜9 まで変化させ，各々の範囲でランダムに生成した透かしを 100 回ずつ埋め込み，平均プログラムサイズを測定した．透かし埋め込み時間も同様に測定した．測定には，コンパイラに `gcc 3.2`，オプションに`-O`，プログラムサイズの取得に `readelf` コマンド，時間の取得には `getrusage` 関数を用いた．使用した計算機は CPU が Athlon XP 1700+，主記憶 512 MB，OS は RedHat Linux 8.0 である．

4　結果

オブジェクトサイズの増加は，図 3 に示した通りである．Radix−k 法が最も増分が小さく，9 桁の透かしを埋め込んだ場合で約 670 byte 増である．同様に，数え上げ法は約 2600 byte，加算数え上げ法は約 2400 byte の増である．

透かし部分だけを考えてみる．文献 [1] では Radix−k 法の 1 ノードは Java の bytecode で 24 byte で構成でき，2040 bit の透かしを埋め込むときの静的ビットレートは約 0.33 bit/byte と述べられている．静的ビットレートとは，透かしのプログラムサイズあたり何ビット透かしを埋め込めるかの割合である．本研究の実装では，1 ノードに約 50 byte 使用した．30 bit 埋め込む場合の静的ビットレートは約 0.067 bit/byte である．2040 bit では約 0.16 bit/byte である．ノードの数に応じて埋め込める数が指数的に増加するためと考えられる．

数え上げ法は，透かしを認識する部分が大部分を占めた．加算数え上げ法は，必要となる有向木の数が増えていくため，サイズの増加が徐々に急になっていく．サイズ増分の大半は透かし部分である．

透かし埋め込み時間の測定結果を図 4 に示す．Radix−k 法は埋め込むまでの時間が他 2 種よりも短い．グラフの構造が単純であるためと考えられる．数え上げ法は，6 桁以上の埋め込みで時間が急激に増大する．大きな透かしを埋め込む場合は 6 桁未満で区切り，複数の有向木に割り当てることが望ましい．加算数え上げ法は，6 桁以上の埋め込みでは数え上げ法よりも埋め込み時間が短い．

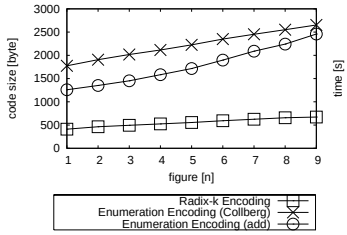


図 3: オブジェクトサイズ増分

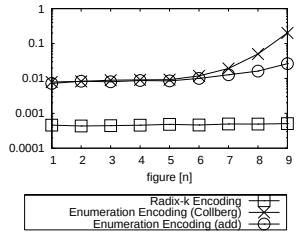


図 4: 埋め込み時間

5　おわりに

文献 [1] では，動的ビットレートの見積も示されている．動的ビットレートとは，グラフを構成する 1 word あたり何ビット透かしを埋め込めるかの割合である．本研究ではグラフを構成するポインタの数が word 数に相当し，Radix−k 法は 2 word，数え上げ法と加算数え上げ法は 1 word で 1 ノードを構成する．

先行研究 [1] では，Radix−k 法で 2040 bit を埋め込む場合，動的ビットレートが 4 bit/word と述べられている．本研究の実装においても 2040 bit 埋め込んだ場合は 4 bit/word あった．30 bit の埋め込みでは 1.58 bit/word であった．同様に，数え上げ法では 1024 bit を埋め込む場合に 1.56 bit/word [1] とあるが，数え上げ法で 1024 bit の透かしを埋め込むと現実的な時間に終了しないと予想される (図 4 より)．大きな透かしを埋め込む場合は，埋め込み時間の観点から言って，透かしを複数のグラフに分割して埋め込む方が現実的と考えられる．本研究の実装では，数え上げ法で 30 bit 埋め込んだとき，動的ビットレート 1.15 bit/word であった．より大きな動的ビットレートが必要な場合は，埋め込み単位を大きくとればよい．ただし埋め込み単位を大きくすると，埋め込み時間は増加する．

加算数え上げ法では，30 bit 埋め込んだとき，動的ビットレート 0.37 bit/word であった．

参考文献

[1] Collberg, C. and Thomborson, C.: Software Watermarking: Models and Dynamic Embeddings, Proc. 26th Symp. Principles of Programming Languages (POPL ’99), pp. 311−324 (1999).

[2] Knuth, D.E.: 基本算法/情報構造, サイエンス社, pp. 174−187 (1978).