Encoding Numbers as Reducible Permutation Graphs for Software Watermarking

Maria Chroni and Stavros D. Nikolopoulos

Department of Computer Science, University of Ioannina, GR-45110, Ioannina, Greece.

Emails: {mchroni, stavros}@cs.uoi.gr

Abstract Based on the fact that a watermark number \( w \) can be efficiently encoded as self-inverting permutation \( m^* \), we present an efficient encoding algorithm which encodes a self-inverting permutation \( m^* \) as a reducible flow-graph \( F[m^*] \) and the corresponding decoding algorithm. Our codec algorithms have very low time and space complexity and the flow-graph \( F[m^*] \) incorporates important structural properties which cause it resilience to attacks [3].

Encode Self-inverting Permutations as Reducible Permutation Graphs

In [1] we introduced the notion of bitonic permutations and we presented two algorithms, for encoding an integer \( w \) into a self-inverting permutation (SiP) \( m^* \) and extracting it from \( m^* \) (see also [2]). In this section, we present efficient algorithms for encoding a SiP \( m^* \) into a reducible permutation graph \( F[m^*] \) and extracting \( m^* \) from \( F[m^*] \).

The proposed encoding algorithm, which we call Encode_SIP-to-RPG, takes as input the SiP \( m^* \) of length \( n \) and constructs a reducible permutation flow-graph \( F[m^*] \) by using certain properties of the decreasing subsequences of \( m^* \). The algorithm takes \( O(n) \) time and requires \( O(n) \) space; it works on two phases:

(I) it first computes the decreasing subsequences \( S_1, S_2, \ldots, S_k \) of the self-inverting permutation \( m^* \) and, then

(II) it constructs a directed graph \( F[m^*] \) on \( n+2 \) nodes using the next relation on the elements of the decreasing subsequences \( S_1, S_2, \ldots, S_k \); see figure 1.

Next, we present the decoding algorithm Decode_RPG-to-SIP, which takes as input a flow-graph \( F[m^*] \) and produces the SiP \( m^* \); it works as follows (sketch):

- Delete the directed edges \( (v_{i+1}, v_i) \) from the graph \( F[m^*] \), and the node \( t \);
- Flip all the remaining directed edges of the graph \( F[m^*] \); the resulting graph is a tree \( T[m^*] \) rooted at node \( s \);
- Compute the pairs \( P_1, P_2, \ldots, P_k \); Return the permutation \( m^* \); see figure 1;
Fig. 1. The main structures used by Encode_SIP-to-RPG and Decode_RPG-to-SIP algorithms.

The algorithm uses binary search to insert the elements of $m^*$ in the queues $Q_1$, $Q_2$, ..., $Q_k$ and requires $O(n \log n)$ time. The sequences $S_1, S_2, ..., S_k$ of $m^*$ can also be commutated in $O(n)$ time using counting sort. Thus, we have:

**Theorem 1.** Let $m^*$ be a self-inverting permutation of length $n$. The algorithm Encode_SIP-to-RPG for encoding the permutation $m^*$ as a reducible permutation flow-graph $F[m^*]$ requires $O(n)$ time and space.

The size of the tree $T[m^*]$ is $O(n)$ since the input graph $F[m^*]$ has $O(n)$ nodes. Based on the structure of the tree $T[m^*]$ we can compute the pairs $P_1, P_2, ..., P_k$ in $O(n)$ time using $O(n)$ space. Thus, we obtain the following result.

**Theorem 2.** Let $F[m^*]$ be a reducible permutation flow-graph of size $O(n)$ produced by the algorithm Encode_SIP-to-RPG. The algorithm Decode_RPG-to-SIP decodes the flow-graph $F[m^*]$ in $O(n)$ time and space.

References

