MCC-HBFT: Algorithms and complementary results

Vitor Hugo Galhardo Moia, Marco Aurélio Amaral Henriques

[1]School of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP)
Campinas, SP, Brasil 13083-852

vhgmoia,marco@dca.fee.unicamp.br

$July\ 5^{th}, 2018$

1. Algorithms: Preparation and Operational phases

input :

- Database of fingerprint templates (T_{all}): path, ID and classes (c_1 and c_2).
- Set of hash functions $F = \{f_{H1}, f_{H2}, ..., f_{Hl}\}$.
- Number of maximum (H_{max}) and minimum (H_{min}) hash functions.
- Number of MCC cylinder sections (s).

output: $HBFT$ structure

Procedure `PreparationPhase` (T_{all}: *list <string, int, char, char>*) : $HBFT$ *struct*
    $HBFT$ = InitializeHBFTstructure() //*Create s BF-trees*
    T_{all} = Group(T_{all}) //*Group according fingerprint classes*
    foreach bf_{tree} **in** $HBFT$ **do**
        //*Insert all database templates*
        foreach T **in** T_{all} **do**
            Extract features of T with hash functions set F and store in f_h.
            //Each feature is composed by H_{max} hashes
            Insert T in one empty BF and all its parents (BF_{set})
            foreach bf **in** BF_{set} **do**
                foreach $feature$ **in** F_h **do**
                    foreach $hash$ **in** $feature.hashes$ **do**
                        if *IsInBF(bf, $hash$) = True* **then**
                            AddHashBF(bf, $hash$)
                        end
                    end
                end
                Set bf classes with c_1 and c_2
                Create and store in the last level BF the attributes of T
            end
        end
    end
    return $HBFT$

Algorithm 1: HBFT Preparation Phase

input :

- Fingerprint template T and its classes (c_1 and c_2).
- $HBFT$ structure with all database fingerprints inserted.
- Set of hash functions $F = \{f_{H1}, f_{H2}, ..., f_{Hl}\}$.
- Number of maximum (H_{max}) and minimum (H_{min}) hash functions.

output: A candidates List=$\{(ID_i, s_i)\}$, with template ID and match score.

```
Procedure Find(bf_tree: Bloom_Filter_Tree, f_h: Hash_Struct, c_1: char, c_2: char, p: int) : list <string, float>
    if IsLeaf(bf_tree, p) = True then
        continue ← 0, hit_features ← 0, hits ← 0
        if (bf_tree[p].class_1 = c_1 OR bf_tree[p].class_1 = c_2) then
            continue ← 1
        else if (bf_tree[p].class_2 = c_1 OR bf_tree[p].class_2 = c_2) then
            continue ← 1
        end
        if continue = 0 then
            return NULL
        end
        foreach feature in F_h do
            hit_hashes ← 0
            foreach hash in feature.hashes do
                if IsInBF(bf_tree[p], hash) = True then
                    hit_hashes ++
                end
            end
            if hit_hashes ≥ H_min then
                foreach hash in feature.hashes do
                    //Compute the Compatible Function
                    if Comp_Func(feature.xyθ, bf_tree[p].xyθ) = True then
                        hit_features ++
                    end
                end
                if hit_features ≥ H_min then
                    hits ← hits + hit_hashes
                end
            end
        end
        s ←
          hits / ((bf_tree[p].num_features + f_h.num_features) · H_max) / 2
        list_temp[i] ← (bf_tree[p].ID, s)
        i ++
    else
        //Looking for feature set f_h in current BF
        if Match(bf_tree[p], f_h, c_1, c_2) = True then
            //Looking for feature set f_h in BF children
            Find(bf_tree, f_h, c_1, c_2, left(p))
            Find(bf_tree, f_h, c_1, c_2, right(p))
        end
    end
    return list
```

```
Procedure Match(bf: Bloom_Filter, f_h: Hash_Struct, c_1: char, c_2: char) : boolean
    continue=0;
    if (bf.class_1 = c_1 OR bf.class_1 = c_2) then
        continue=1;
    else if (bf.class_2 = c_1 OR bf.class_2 = c_2) then
        continue=1;
    end
    if continue = 0 then
        return False
    end
    hit_features ← 0;
    foreach feature in F_h do
        hit_hashes = 0;
        foreach hash in feature.hashes do
            if IsInBF(bf, hash) = True then
                hit_hashes ++;
            end
        end
        if hit_hashes ≥ H_min then
            hit_features ++
        end
        if hit_hashes = H_min then
            return True
        end
    end
    return False
Procedure Main(HBFT: HBFT struct, T: string, c_1: char, c_2: char)
    List = InitializeListOfResults();
    foreach bf_tree in HBFT do
        tempList ← CreateTemporaryResultList();
        f_h ← ExtractFeatures(T, F) ;        /* features are composed by H_max hashes */
        tempList ← Find(bf_tree, f_h, c_1, c_2, 1) ;  /* Starting search in root BF */
        List ← UpdateListResults(List, tempList);
    end
    //Sorting list according to candidate's score;
    List ← SortListResults(List);
    Print(List);
```

Algorithm 2: HBFT Preparation Phase

2. Complementary results

In this section, we present all results got from MCC-HBFT in the public fingerprint databases NIST (DB4) and FVC (2002DB1, 2002DB3, and 2004DB1).

For each database, we present the results of MCC-HBFT in three different settings: low, mid, and high version costs. Since we ran each trial 10 times, we also present for each setting the worst, average, and best case scenario.

2.1. FVC2002DB1

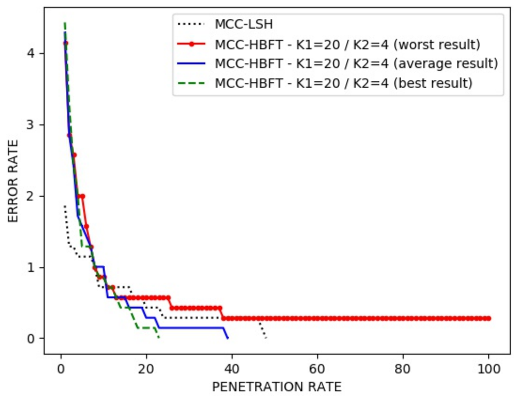

Figure 1. Performance evaluation on FVC2002 DB1: The worst, average, and best case scenarios with a low-cost setting ($k_{max} = 20$ / $k_{min} = 4$)

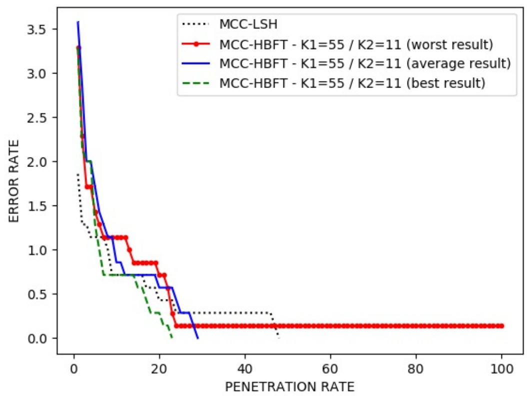

Figure 2. Performance evaluation on FVC2002 DB1: The worst, average, and best case scenarios with a mid-cost setting ($k_{max} = 55$ / $k_{min} = 11$)

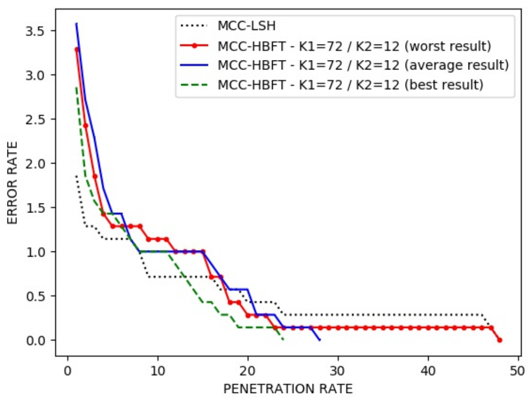

Figure 3. Performance evaluation on FVC2002 DB1: The worst, average, and best case scenarios with a high-cost setting ($k_{max} = 72$ / $k_{min} = 12$)

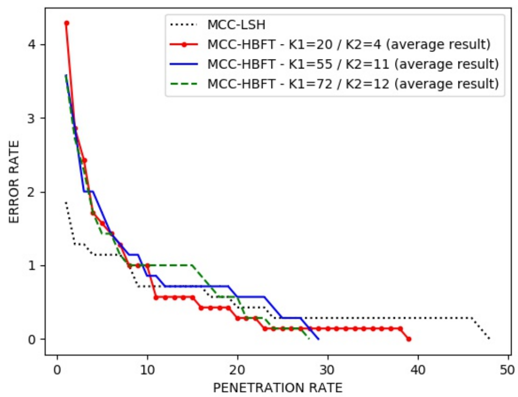

Figure 4. Performance evaluation on FVC2002 DB1: Average case scenario of three different MCC-HBFT versions

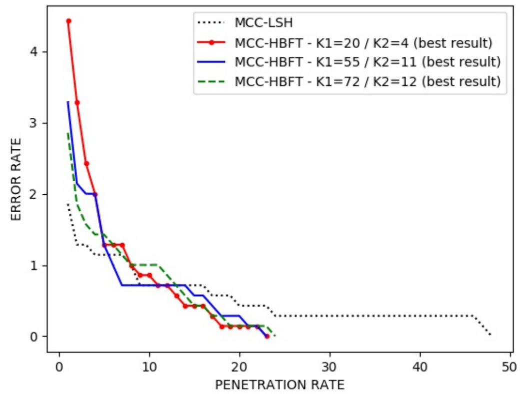

Figure 5. Performance evaluation on FVC2002 DB1: Best case scenario of three different MCC-HBFT versions

2.2. FVC2002DB3

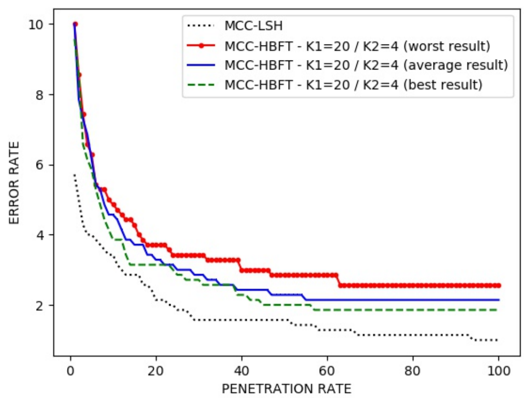

Figure 6. Performance evaluation on FVC2002 DB3: The worst, average, and best case scenarios with a low-cost setting ($k_{max} = 20$ / $k_{min} = 4$)

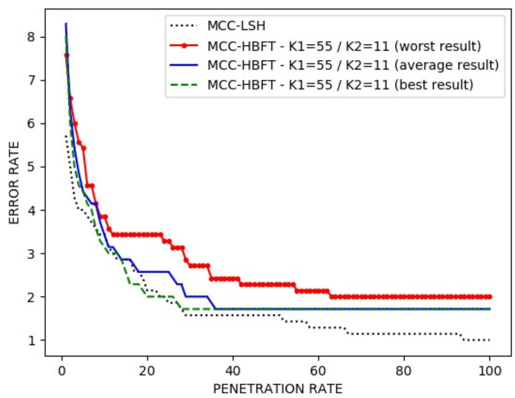

Figure 7. Performance evaluation on FVC2002 DB3: The worst, average, and best case scenarios with a mid-cost setting ($k_{max} = 55$ / $k_{min} = 11$)

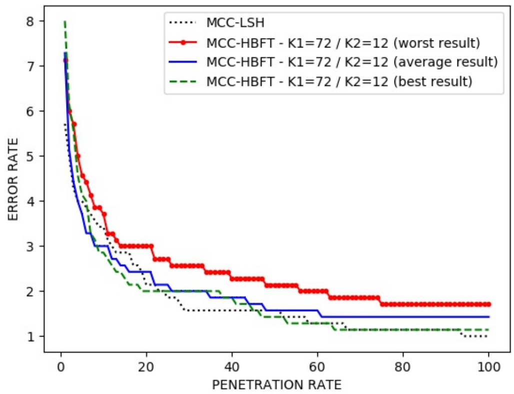

Figure 8. Performance evaluation on FVC2002 DB3: The worst, average, and best case scenarios with a high-cost setting ($k_{max} = 72$ / $k_{min} = 12$)

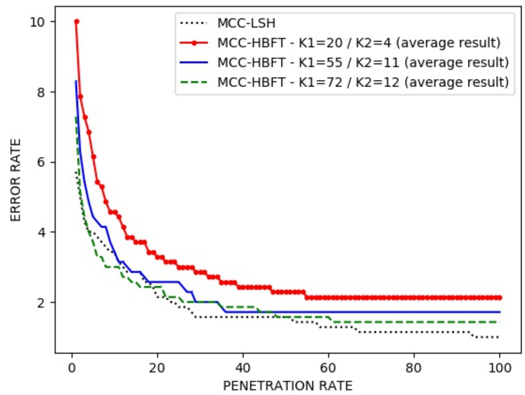

Figure 9. Performance evaluation on FVC2002 DB3: Average case scenario of three different MCC-HBFT versions

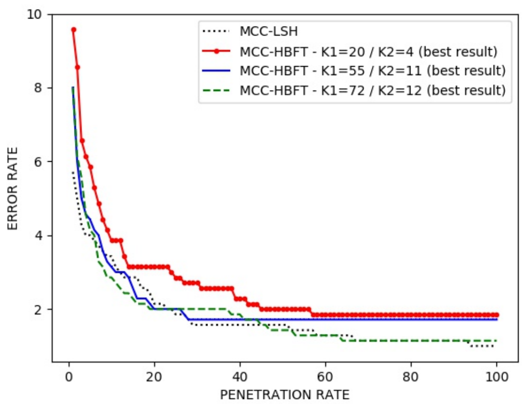

Figure 10. Performance evaluation on FVC2002 DB3: Best case scenario of three different MCC-HBFT versions

2.3. FVC2004DB1

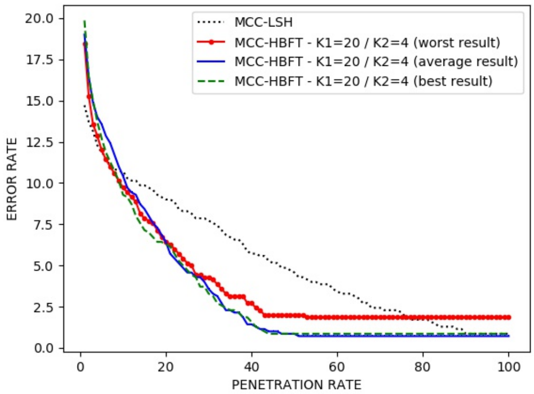

Figure 11. Performance evaluation on FVC2004 DB1: The worst, average, and best case scenarios with a low-cost setting ($k_{max} = 20$ / $k_{min} = 4$)

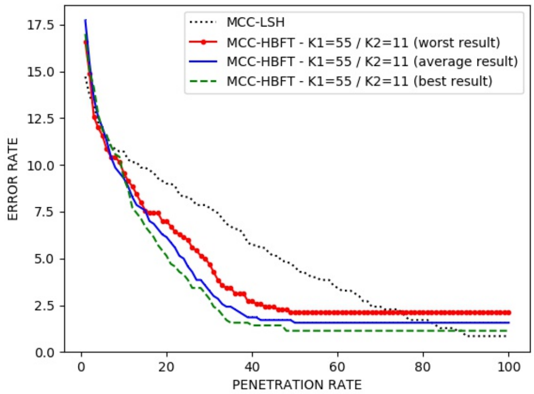

Figure 12. Performance evaluation on FVC2004 DB1: The worst, average, and best case scenarios with a mid-cost setting ($k_{max} = 55$ / $k_{min} = 11$)

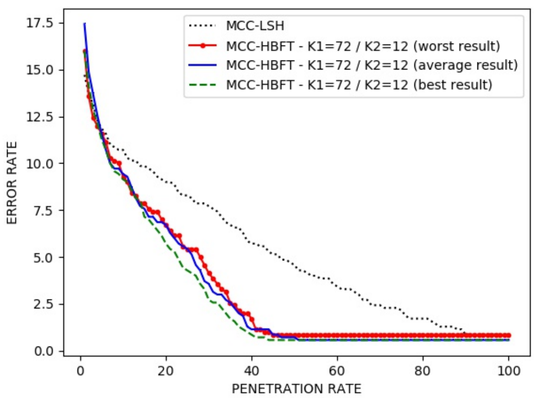

Figure 13. Performance evaluation on FVC2004 DB1: The worst, average, and best case scenarios with a high-cost setting ($k_{max} = 72$ / $k_{min} = 12$)

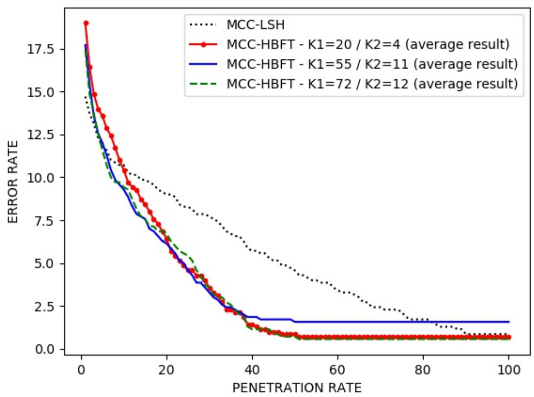

Figure 14. Performance evaluation on FVC2004 DB1: Average case scenario of three different MCC-HBFT versions

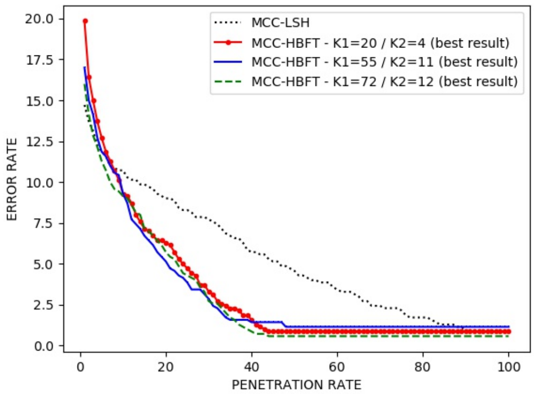

Figure 15. Performance evaluation on FVC2004 DB1: Best case scenario of three different MCC-HBFT versions

2.4. NIST DB4

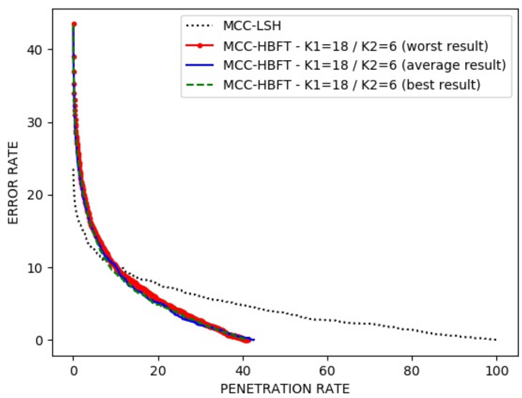

Figure 16. Performance evaluation on NIST DB4: The worst, average, and best case scenarios with a low-cost setting ($k_{max} = 18$ / $k_{min} = 6$)

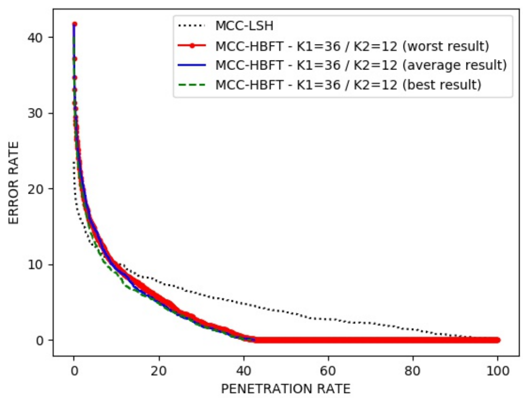

Figure 17. Performance evaluation on NIST DB4: The worst, average, and best case scenarios with a mid-cost setting ($k_{max} = 36$ / $k_{min} = 12$)

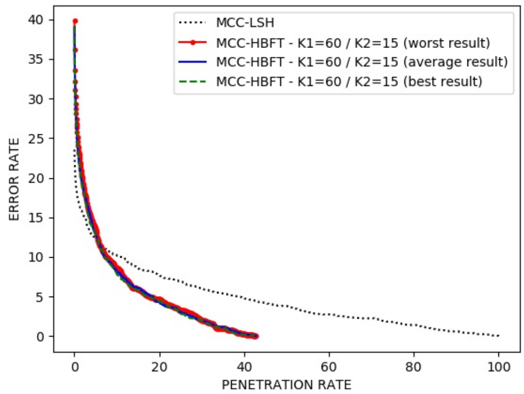

Figure 18. Performance evaluation on NIST DB4: The worst, average, and best case scenarios with a high-cost setting ($k_{max} = 60$ / $k_{min} = 15$)

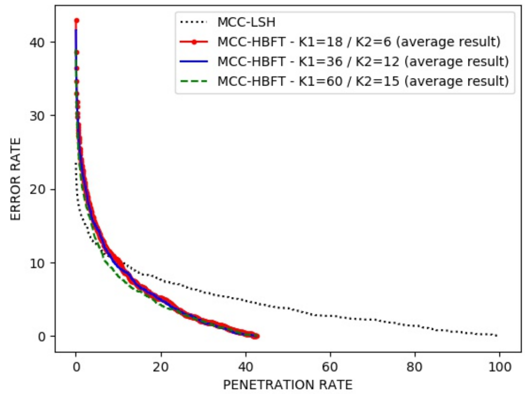

Figure 19. Performance evaluation on NIST DB4: Average case scenario of three different MCC-HBFT versions

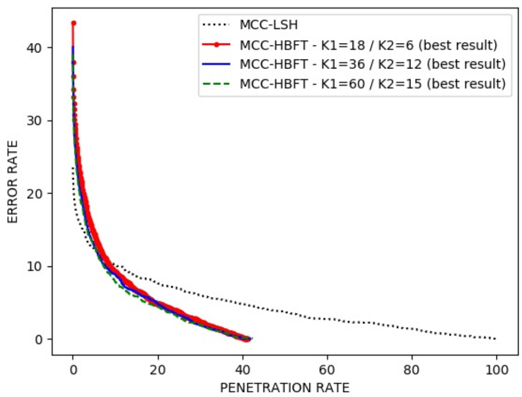

Figure 20. Performance evaluation on NIST DB4: Best case scenario of three different MCC-HBFT versions