Perfect Hashing via Prime Factorization

Hutter RNN Project

2026-01-31

1 The Problem

Given a product pattern $(e_1, e_2) \in E \times E$, we want a perfect hash:

$$h : E \times E \to \mathbb{N}$$

such that h is injective (no collisions).

2 Construction

2.1 Step 1: Project E onto N via Primes

Assign each event $e \in E$ to a unique prime p_e:

$$\pi : E \to \text{Primes} \subset \mathbb{N}$$

For bytes ($|E| = 256$), use the first 256 primes:

$$\pi(0) = 2 \tag{1}$$
$$\pi(1) = 3 \tag{2}$$
$$\pi(2) = 5 \tag{3}$$
$$\vdots \tag{4}$$
$$\pi(255) = 1619 \quad \text{(256th prime)} \tag{5}$$

2.2 Step 2: Factor the Joint Event

For a product pattern (e_1, e_2), the hash is:

$$h(e_1, e_2) = p_{e_1} \cdot p_{e_2}$$

Key property: This is a perfect hash because prime factorization is unique. Given $h(e_1, e_2) = n$, we can recover (e_1, e_2) by factoring n.

2.3 Step 3: Add Log Support

The log support for the joint event decomposes:

$$t_{(e_1,e_2)} = t_{e_1} + t_{e_2|e_1}$$

In prime-space, this becomes:

$$\log h(e_1, e_2) = \log p_{e_1} + \log p_{e_2}$$

The hash value encodes both:

- **Identity:** Which events (via factorization)
- **Support:** How much evidence (via log of hash)

2.4 Step 4: Apply Event Spaces

Factor through ES:

$$e_1 \in ES_{e_1} \times \text{within}_{e_1} \tag{6}$$
$$e_2 \in ES_{e_2} \times \text{within}_{e_2} \tag{7}$$

The ES-level hash uses ES-indexed primes:

$$h_{ES}(es_1, es_2) = p_{es_1} \cdot p_{es_2}$$

With only 5 ESs, we use primes $\{2, 3, 5, 7, 11\}$.
The full hash factors:

$$h(e_1, e_2) = h_{ES}(es_1, es_2) \cdot h_{\text{within}}(w_1, w_2)$$

3 The Two-Ring Visualization

3.1 Structure

Draw two concentric rings:

- **Outer ring:** Source events e_1 (or ES_1)
- **Inner ring:** Target events e_2 (or ES_2)

An arc from outer to inner represents a pattern (e_1, e_2).

3.2 Why Two Rings, Not One

One ring (our previous pattern-rings) shows $e \to e$ as arcs on a single circle.
Two rings separate source and target:

- Clearer visualization of asymmetry
- No arc crossings for "diagonal" patterns
- Natural for product structure: outer = first factor, inner = second

3.3 Log Support as Arc Weight

Arc thickness or color encodes:

$$\text{weight}(e_1 \to e_2) = t_{(e_1,e_2)} = \log \text{count}(e_1, e_2)$$

4 The Perfect Hash Function

Combining everything:

1. **Input:** Pattern (e_1, e_2)
2. **Factor:** $(es_1, w_1), (es_2, w_2)$
3. **Prime encode:**
$$h = p_{es_1} \cdot p_{es_2} \cdot q_{w_1} \cdot q_{w_2}$$
where p are ES-primes and q are within-ES primes.
4. **Output:** Unique integer $h \in \mathbb{N}$

4.1 Decoding

Given hash h:

1. Factor h into primes
2. Separate ES-primes from within-primes
3. Recover (es_1, w_1) and (es_2, w_2)
4. Reconstruct e_1, e_2

5 Collision Probability

The hash is **probabilistically perfect**: collision probability depends on bits allocated.

5.1 Bit Allocation

Given k bits for the hash, partition into:

- k_1 bits for source event e_1
- k_2 bits for target event e_2
- Constraint: $k_1 + k_2 \leq k$

5.2 Collision Probability per Bit

For log-support values with noise ϵ:

$$P(\text{collision in 1 bit}) = P(|t_1 - t_2| < \epsilon) \approx \epsilon \cdot p(t)$$

where $p(t)$ is the density of log-support values.

5.3 Optimal Bit Assignment

Assign bits to minimize total collision probability:

$$k_1^* = \arg\min_{k_1} \left[P(\text{collision}_1 | k_1) + P(\text{collision}_2 | k - k_1) \right]$$

For uniform log-support: split evenly $k_1 = k_2 = k/2$.
For skewed distributions: allocate more bits to higher-entropy factor.

5.4 The Noise Floor

With finite data, log-support estimates have variance:

$$\text{Var}(\hat{t}_e) \approx \frac{1}{n_e}$$

This sets a floor on distinguishability. Two events with counts n_1, n_2 are distinguishable if:

$$|t_1 - t_2| > \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}$$

The "perfect" hash becomes perfect in the limit $n \to \infty$.

6 Summary

Component	Representation
Event e	Prime p_e
Joint (e_1, e_2)	Product $p_{e_1} \cdot p_{e_2}$
Log support	$\log(p_{e_1} \cdot p_{e_2})$
ES factorization	Separate prime families
Visualization	Two joined rings

The prime factorization gives us:

- Perfect hashing (unique encoding)
- Additive log support (multiplication $\to$ addition in log)
- Factored structure (ES $\times$ within)
- Reversibility (can decode by factoring)