

Fast Evaluation of Union-Intersection Expressions

Philip Bille

Anna Pagh

Rasmus Pagh

IT University of Copenhagen

Data Structures for Intersection Queries

- Preprocess a collection of sets S_1, \dots, S_m independently into a representation that supports intersection queries of the form $S_i \cap S_j, 1 \leq i, j \leq m$.
- Application: Boolean AND-queries in search engines.
 - For each word store the set of documents containing the word.
 - To search for documents that contains words x *and* y compute the intersection of the corresponding document sets.
- Generalizes to arbitrary expressions over set collection involving intersection, union, and difference.

Previous Comparison-Based Results

- Query: $S_1 \cap S_2$
- Classical solution:
 - Represent sets as sorted lists.
 - Query by merging and reporting duplicates: $O(|S_1| + |S_2|)$ time.
- Special cases with faster solutions:
 - When $S_1 \ll S_2$: $O(|S_1| \log(1 + \frac{S_1}{S_2}))$ time [HL1972].
 - When $S_1 \cap S_2$ consists of few sublists from S_1 and S_2 [DLM2000, BK2002]. (adaptive algorithms).
- Generalizations to more complicated expressions involving intersections and unions [CFM2005].

Previous Non-Comparison Based Results

- Fast solution when $S_1 \ll S_2$:
 - Build a hashing-based dictionary for each set.
 - Lookup the elements of S_1 in the dictionary for S_2 : $O(S_1)$ time.
- For very small universes:
 - Represent sets as bitstrings.
 - Compute intersections as a bitwise-AND.

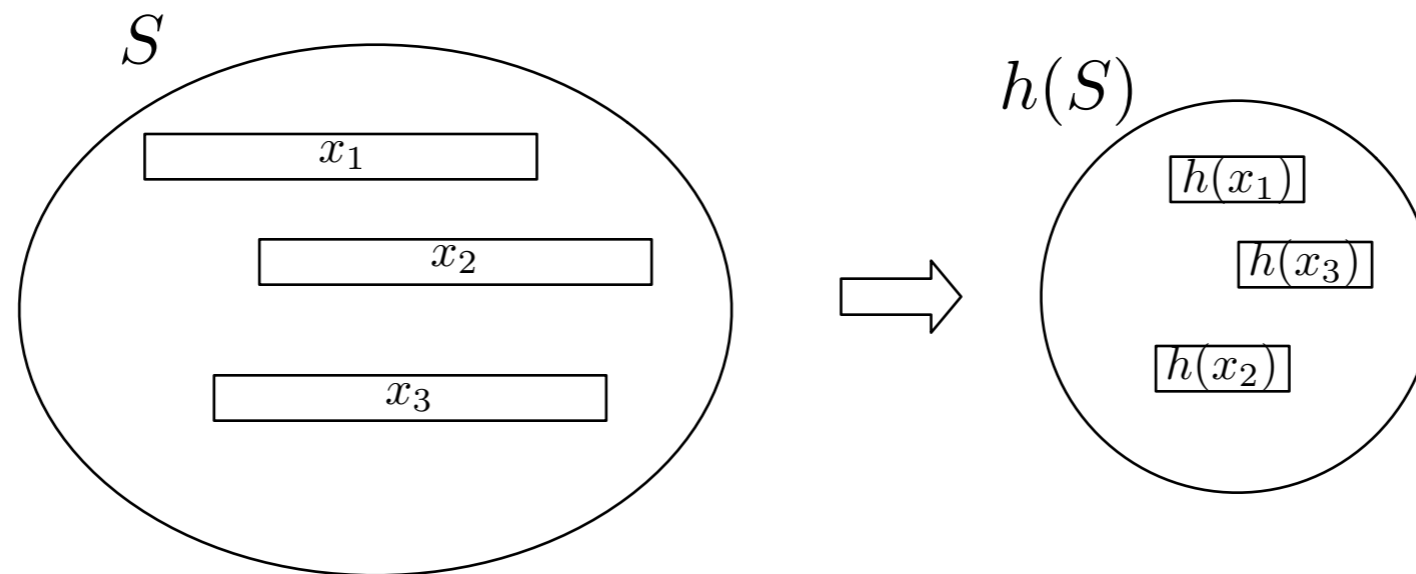
Our Results

- **Theorem:** There is a non-comparison based linear space representation supporting intersection queries $S_1 \cap S_2$ queries in expected time

$$O\left(\frac{(|S_1| + |S_2|) \log^2 w}{w} + \text{occ}\right)$$

- Output-sensitive algorithm.
- For $\text{occ} < (|S_1| + |S_2|)/w$ the algorithm runs in sublinear time.
- All previously known solutions use worst-case linear time even if the intersection is empty.
- We show how to generalize the result to arbitrary union-intersection expressions.
- We give a communication complexity lower bound proving that the result is near optimal.

Approximate Set Representation



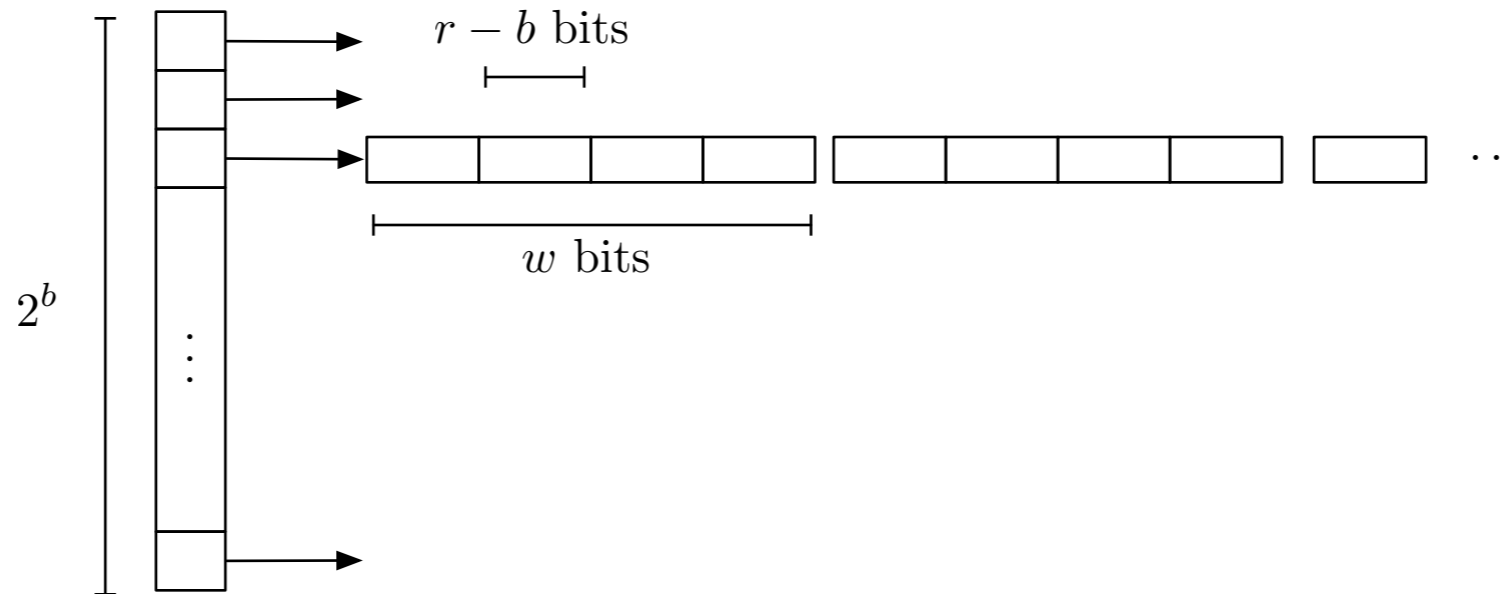
- Represent set $S \subseteq \{0, 1\}^w$ as a set of hash function values $h(S)$.
- $h(S)$ is an *approximate set representation*:
 - If $x \in S$ then $h(x) \in h(S)$.
 - if $x \notin S$ then $h(x) \notin h(S)$ with probability close to 1.

Computing Intersections

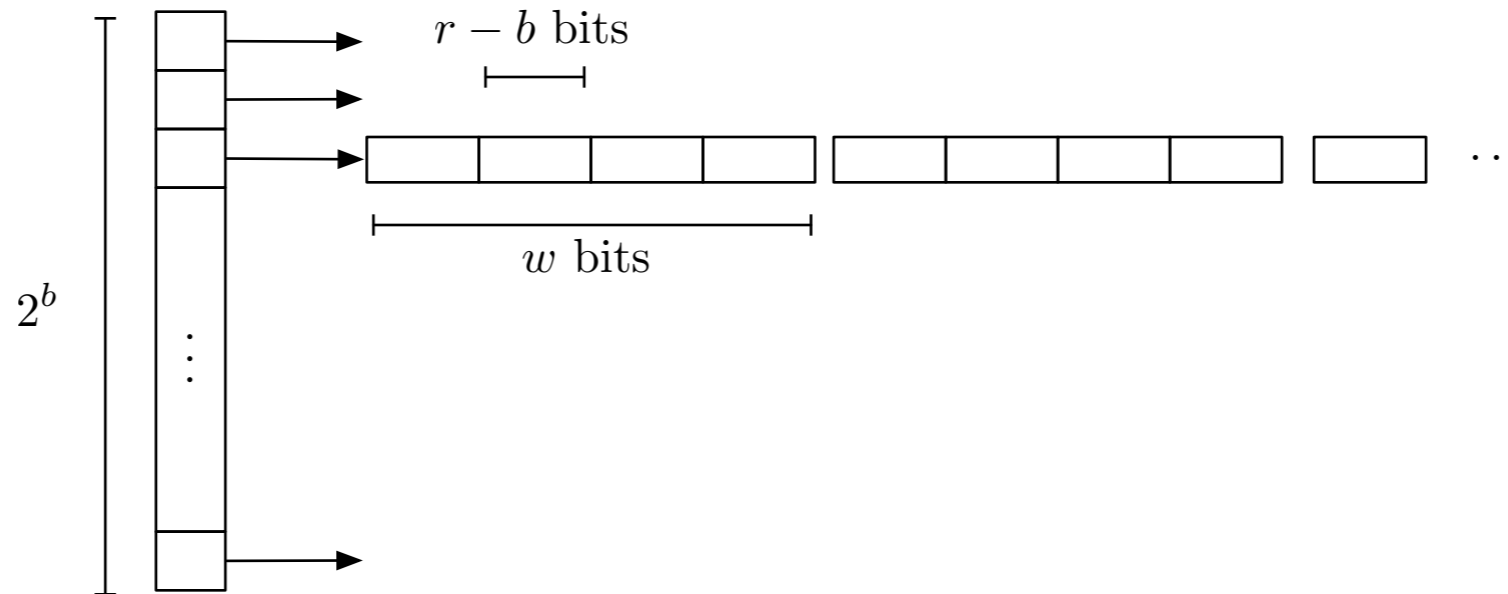
1. Compute intersection of the approximate representations $H = h(S_1) \cap h(S_2)$.
 - We do this in $o(|S_1| + |S_2|)$ time.
2. Compute $S'_1 = \{x \in S_1 \mid h(x) \in H\}$ and $S'_2 = \{x \in S_2 \mid h(x) \in H\}$.
 - With a hash table that allows us to lookup a value $h(x)$ and retrieve all elements with this value this takes $O(|S'_1| + |S'_2|)$ time.
3. Compute and return $S'_1 \cap S'_2$.
 - Idea: If the hash function is suitably chosen, the number of elements to be checked in step 2 is small.

Choosing Hash Functions

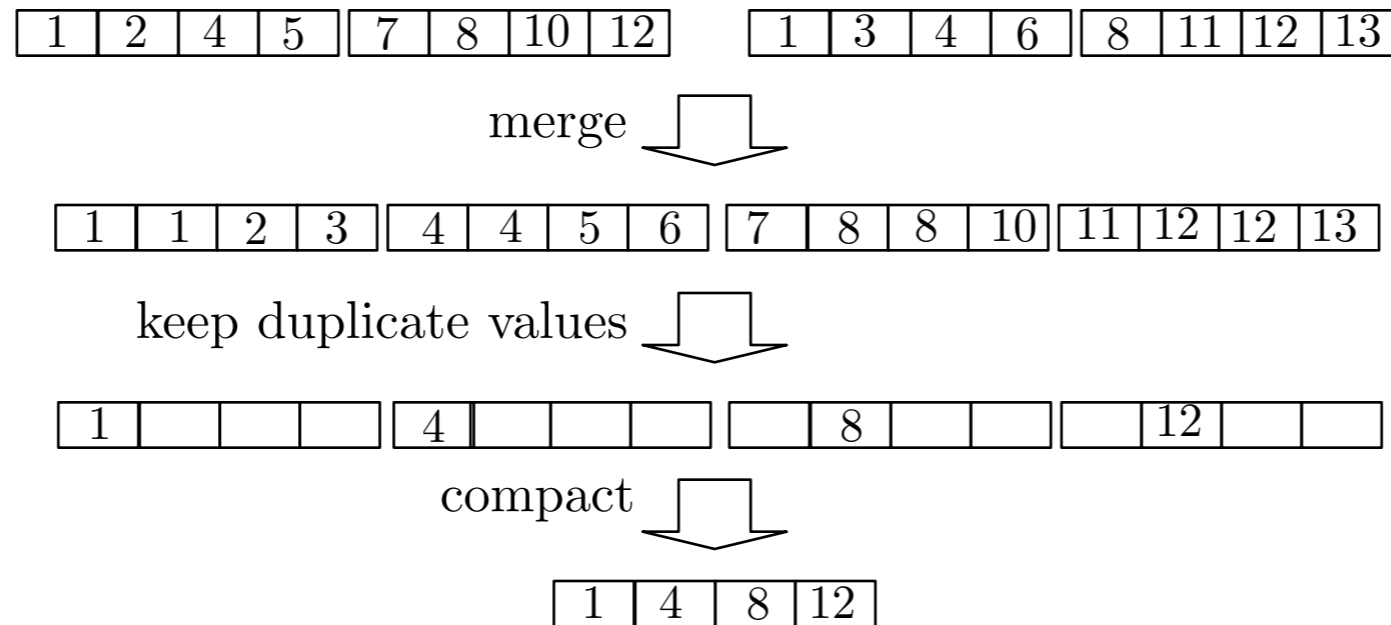
- The number of bits used for the hash values should be:
 - Small enough so that $H = h(S_1) \cap h(S_2)$ can be computed quickly.
 - Large enough to get a significant reduction in the number remaining elements in $S'_1 = \{x \in S_1 \mid h(x) \in H\}$ and $S'_2 = \{x \in S_2 \mid h(x) \in H\}$ so that $S'_1 \cap S'_2$ can be computed quickly.
- Optimal range of hash function depends on the size of input sets.
 - We store S at “multiple resolutions” using hash functions with different ranges.



- We store a set $h_r(S)$ of r -bit hash values as a *bucketed set* for parameter b :
 - Elements with the same b most significant bits are stored in the same bucket.
 - Elements in the same bucket are represented by their $r - b$ least significant bits as a sorted *packed array*.
- We choose b to minimize total space.
 - We can store a sufficient set of resolutions of S in total linear space.
 - $r - b = O(\log w)$.



- Intersection algorithm for bucketed sets:
 - Convert buckets to have a common (suitable chosen) parameter b .
 - Create a new array of size 2^b .
 - Repartition packed arrays among the new buckets.
 - Modify number of bits in packed array representation.
 - Compute intersection among each of the sorted packed arrays.



- **Lemma:** [AH1992, ATNR1995] All of the above operation can be computed in time $O(\log w)$ per word in the packed arrays.

- Total time: $O\left(\frac{(|S_1| + |S_2|) \log w}{w} \cdot \log w\right)$

Our Results

- **Theorem:** There is a non-comparison based linear space representation supporting intersection queries $S_1 \cap S_2$ queries in expected time

$$O\left(\frac{(|S_1| + |S_2|) \log^2 w}{w} + \text{occ}\right)$$

- In the paper:
 - Generalization to arbitrary union-intersection expressions
 - Lower bound
- Open Problem:
 - Can we extend this to set difference?