Time-Memory Trade-offs for Parallel Collision Search Algorithms

Monika Trimoska Sorina Ionica Gilles Dequen

Laboratoire MIS, Université de Picardie Jules Verne

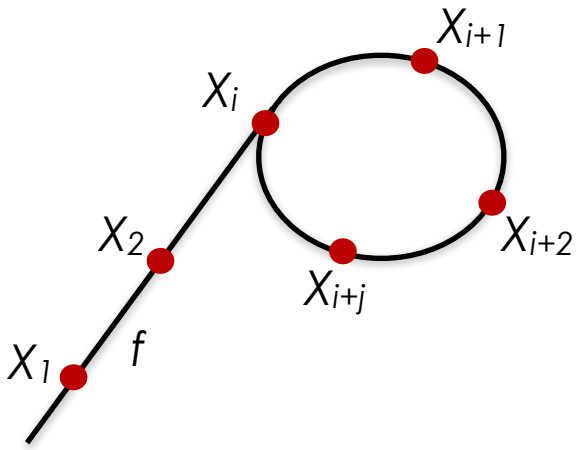
Journées Codage & Cryptographie

11 oct. 2018, Aussois

Collision Search

- Given a function $f : S \rightarrow S$ on a finite set S,
  we call **collision** any pair a,b of elements in S such that $f(a) = f(b)$.

- Pollard's rho method



- Ideally, f is a random mapping.

Collision Search Applications

- One collision application: **discrete logarithm**
  Given a group G of prime order and g a generator of G and $h \in G$, find x such that,

$$g^x = h$$

- Multi-collision application: **Meet-In-The-Middle**
  Attack on the 3-DES with three independent keys.
  In the worst case $\frac{n}{2}$ collisions are generated.

Parallel Collision Search

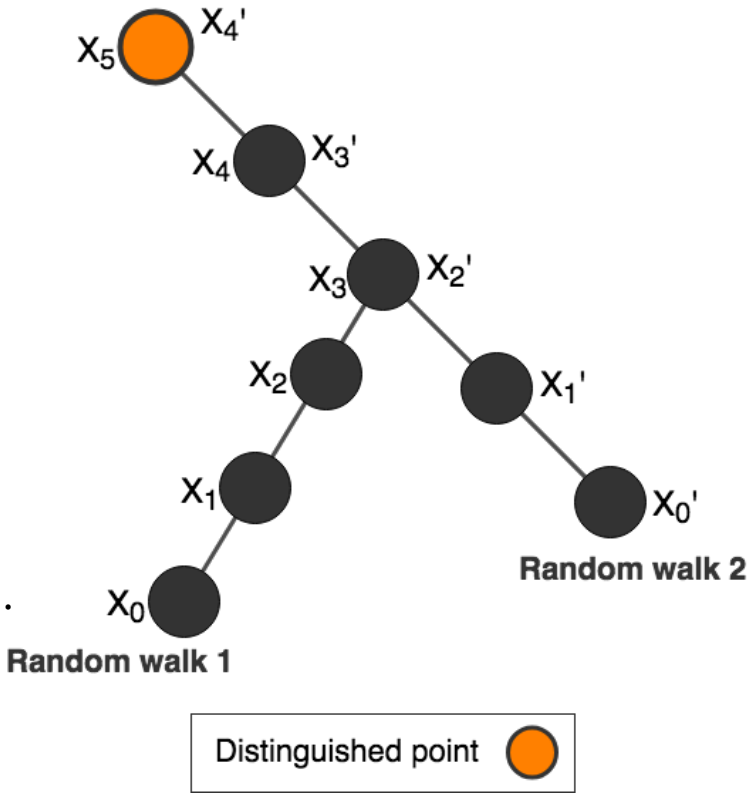
van Oorschot & Wiener, 1996

- Collision : find two different input points that produce the same output point.

- Distinguished points : a set of points having an easily testable property.
  ex. The x-coordinate has 3 trailling zero bits:
  10101101000

- Only distinguished points are stored in memory.

- θ - the proportion of distinguished points in a set S.



Time complexity

Theorem. In the parallel collision search algorithm, the expected running time to find ***m*** collisions with a memory constraint of ***w*** words is:

expected number of iterations needed to find and store ***w*** points

number of collisions found after storing ***w*** points

expected number of iterations needed to find one collision when ***w*** points are stored

$$\frac{1}{L}\left(\frac{w}{\theta}+\left(m-\frac{w^2}{2\theta^2 n}\right)\frac{\theta n}{w}+\frac{2m}{\theta}\right)$$

Time complexity

Corollary. The optimum proportion of distinguished points minimizing the time complexity is

$$\theta = \frac{\sqrt{w^2 + 2nw}}{n}.$$

The running time of the parallel collision search algorithm for finding $\frac{n}{2}$ collisions is bounded by:

$$O\left(\frac{n}{L}\sqrt{1 + \frac{2n}{w}}\right)$$

→ Memory is an important factor in the running time complexity.

Data structure

- Requirements:

→ Space efficient

→ Thread-safe

→ Fast look-up and insertion

- Most commonly used structure:
  Hash table

Data structure

- Requirements:

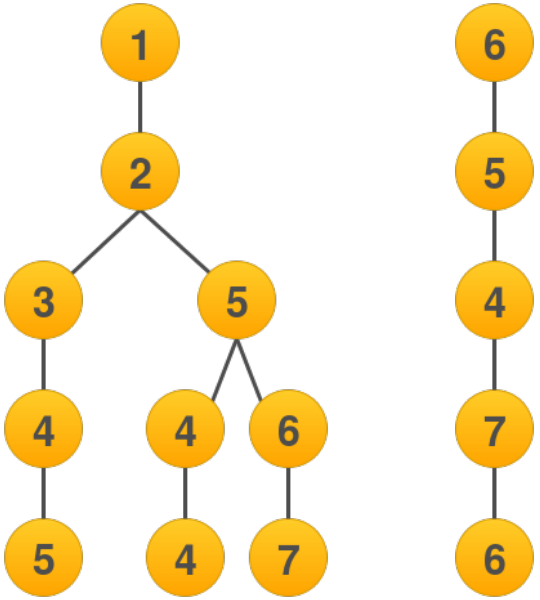
→ Space efficient

→ Thread-safe

→ Fast look-up and insertion

- Most commonly used structure:
  Hash table

- Alternative:
  Radix tree



Exemple of a radix tree holding the set
12345, 12544, 12567, 65476

Data structure

- Requirements:

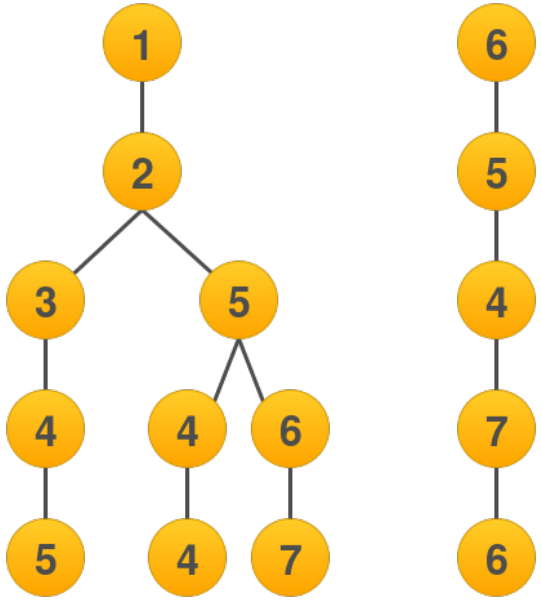
→ Space efficient

→ Thread-safe

→ Fast look-up and insertion ✓

- Most commonly used structure:
  Hash table

- Alternative:
  Radix tree



Exemple of a radix tree holding the set
12345, 12544, 12567, 65476

Data structure

- Requirements:

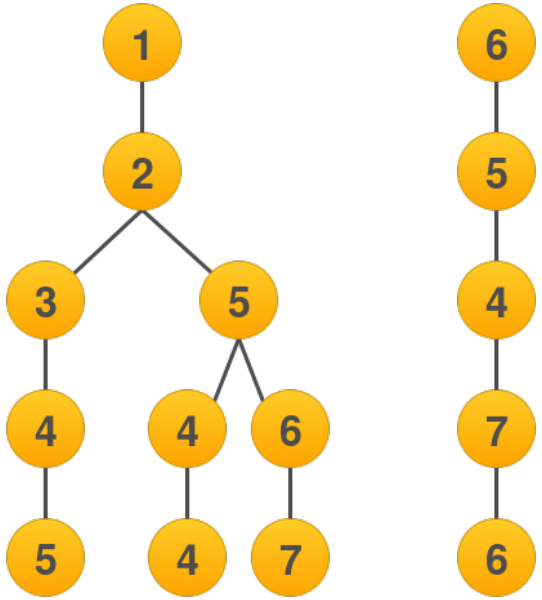
→ Space efficient

→ Thread-safe

→ Fast look-up and insertion

- Most commonly used structure:
  Hash table

- Alternative:
  Radix tree



Exemple of a radix tree holding the set
12345, 12544, 12567, 65476

Data structure

- Requirements:

→ Space efficient

→ Thread-safe

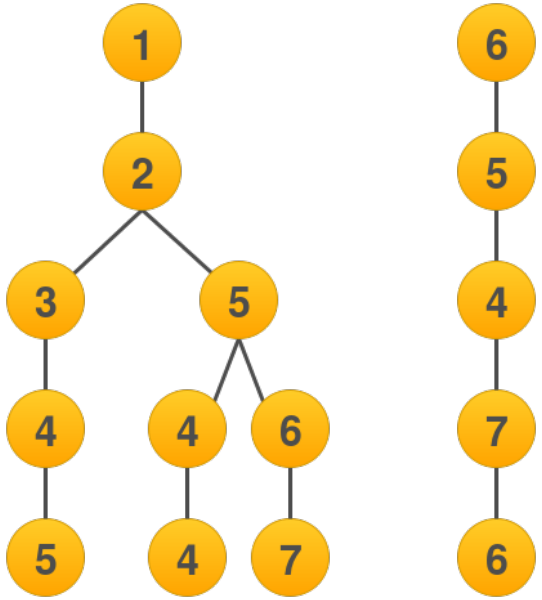
→ Fast look-up and insertion

(+) Common prefixes

(−) Pointers to empty branches

- Most commonly used structure:
  Hash table

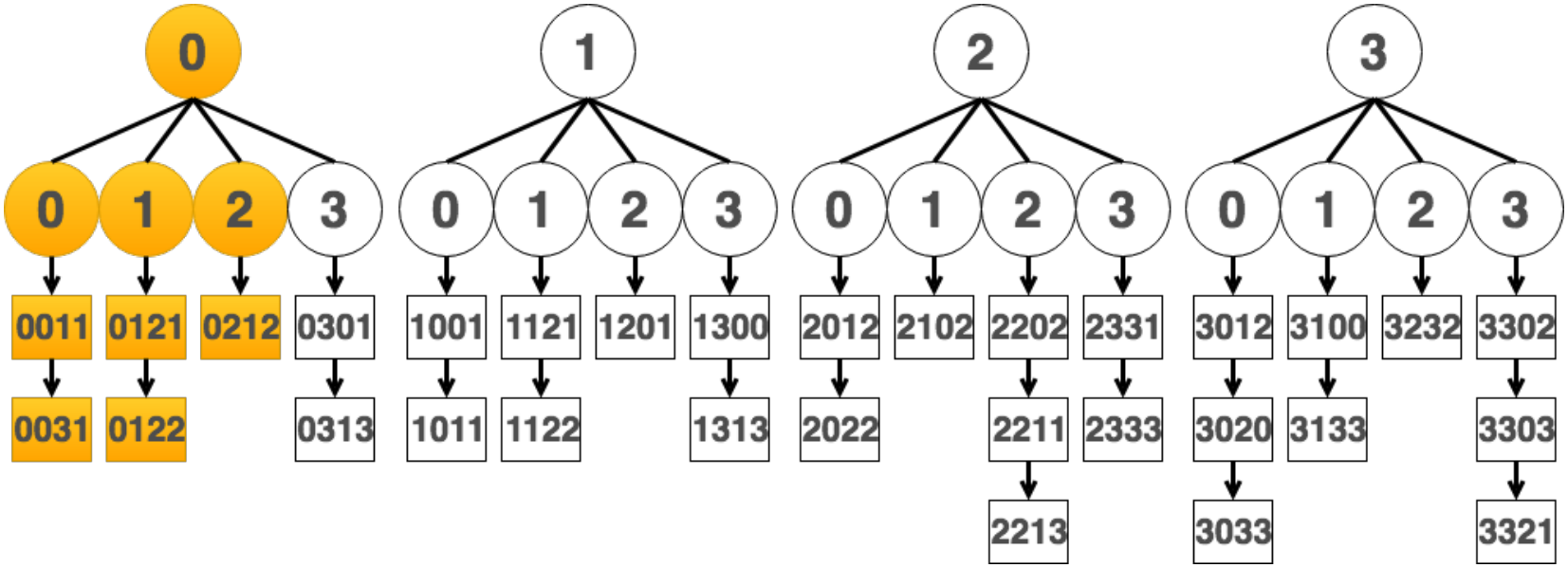
- Alternative:
  Radix tree



Exemple of a radix tree holding the set
12345, 12544, 12567, 65476

Packed Radix-Tree-List

- Construct a radix tree up to certain level
- Add the points to linked lists, each list starting from a leaf on the tree



Exemple of a radix tree holding the set 0011, 0031, 0121, 0122, 0212, etc.

Finding the optimal branching level

Find level l such that

- There are no pending leaves
- Linked lists are as short as possible

Finding the optimal branching level

Find level l such that

- There are no pending leaves
- Linked lists are as short as possible

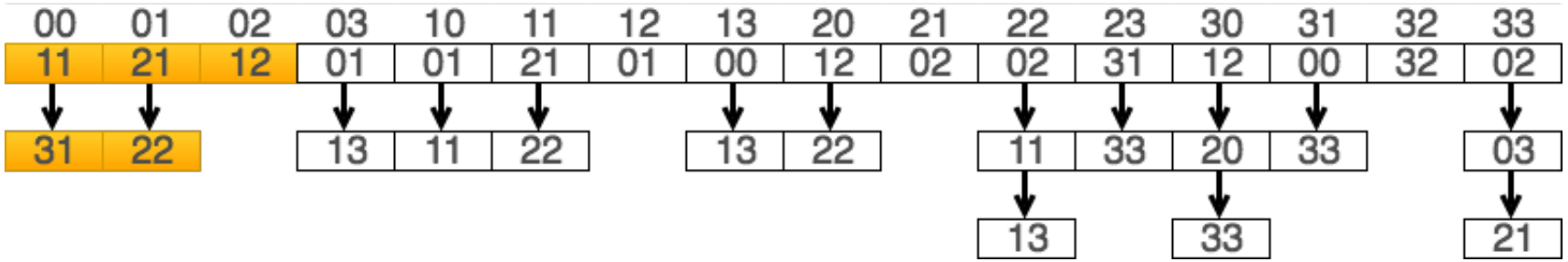
As per the Coupon collector's problem, the optimal level is l such that

$$b^l(\ln b^l + 0.577) \sim K$$

where ***K*** is the estimated number of stored points and ***b*** is the base of their numerical representation.

Implementation

- Collision search in $E(\mathbb{F}_p)$, with p prime, to solve the discrete logarithm problem.
- $\theta \sim \frac{1}{2^{b/4}}$ for a b-bit curve.
- In C, using external libraries GMP and OpenMP.
- 28-core Intel Xeon E5-2640 processor.
- Experimenting with 1 to 28 threads.
- Structure : **Packed** Radix-Tree-List.



Exemple of a radix tree holding the set 0011, 0031, 0121, 0122, 0212, etc.

When memory is limited

Running a multi-collision search while limiting the available memory proves that more storage space yields a faster algorithm.

Collisions	Memory limit	Runtime		Stored Points	
		PRTL	Hash table	PRTL	Hash table
400	10MB	14,3 min	18,8 min	474019	216459
10000	40MB	74,2 min	113,4 min	2104832	867429
50000	100MB	172,5 min	241,6 min	5262727	2169383

Multi-collision search for a 55-bit curve

In conclusion

- Revisited one-collision and multi-collision time complexity
- Showed that memory is an important factor in the running time complexity
- Proposed an alternative memory structure

https://eprint.iacr.org/2017/581