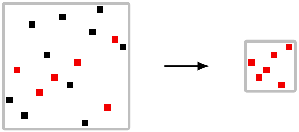
Exhaustive search of permutations with many patterns

Axel Bacher   Michael Engen

April 14, 2020

Outline

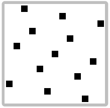
Permutations with many patterns



- How many patterns of size k can a permutation of size n contain?
- What are the optimal permutations like?
- Given n and k, can we construct an optimal permutation?

Universal and prolific permutations

[Bevan–Homberger–Tanner 2017, Engen–Vatter 2020]

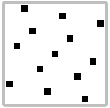


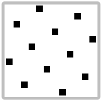
5-universal

- Permutations with $k!$ patterns of size k are called k-universal.
- They exist iff $n \geq L_k$, with $e^{-2}k^2 \leq L_k \leq \left\lceil \frac{k^2+1}{2} \right\rceil$.

Universal and prolific permutations

[Bevan–Homberger–Tanner 2017, Engen–Vatter 2020]

 5-universal

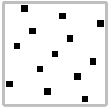
 3-prolific

- Permutations with $k!$ patterns of size k are called k-universal.
- They exist iff $n \geq L_k$, with $e^{-2}k^2 \leq L_k \leq \left\lceil \frac{k^2+1}{2} \right\rceil$.

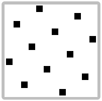
- Permutations with $\binom{n}{k}$ patterns of size $n-k$ are called k-prolific.
- They exist iff $n \geq \lceil k^2/2 + 2k + 1 \rceil$.
- Criterion: $|i-j| + |\sigma_i - \sigma_j| \geq k+2$ for all $i \neq j$.

Universal and prolific permutations

[Bevan–Homberger–Tanner 2017, Engen–Vatter 2020]



5-universal



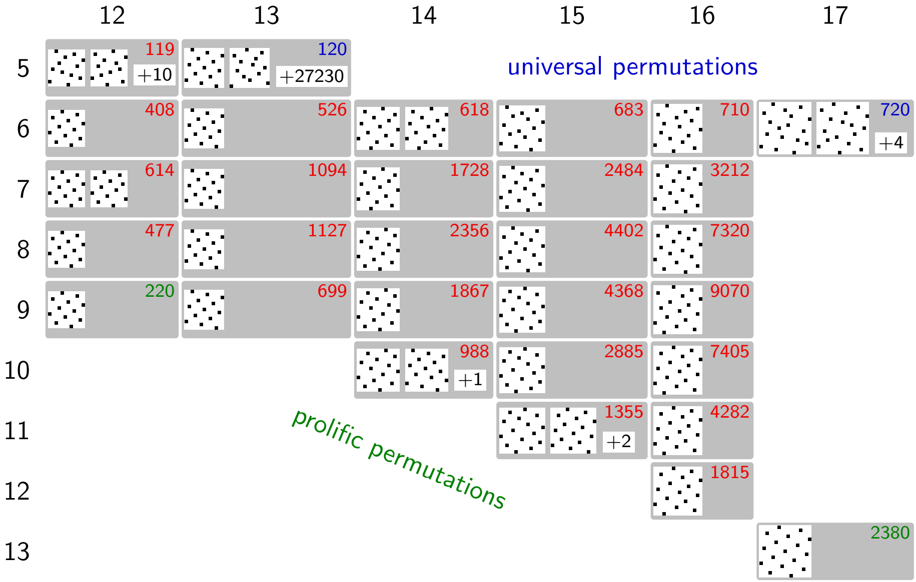
3-prolific

- Permutations with $k!$ patterns of size k are called k-universal.
- They exist iff $n \geq L_k$, with $e^{-2}k^2 \leq L_k \leq \left\lceil \frac{k^2+1}{2} \right\rceil$.

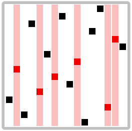
- Permutations with $\binom{n}{k}$ patterns of size $n-k$ are called k-prolific.
- They exist iff $n \geq \lceil k^2/2 + 2k + 1 \rceil$.
- Criterion: $|i-j| + |\sigma_i - \sigma_j| \geq k+2$ for all $i \neq j$.

- When $\Theta(\sqrt{n}) < k < n - \Theta(\sqrt{n})$, there are $< \min\left[k!, \binom{n}{k}\right]$ patterns.

Optimal permutations: experimental results



Ranking patterns

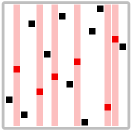


$$\mathrm{rk}(\sigma, S) = 3 \times 120 + 24 + 6 + 2 = 392$$

- We rank patterns based on their inversions:

$$\mathrm{rk}(\sigma, S) = \sum_{i,j \in S,\, i<j,\, \sigma_i > \sigma_j} |S_{>i}|!$$

Ranking patterns



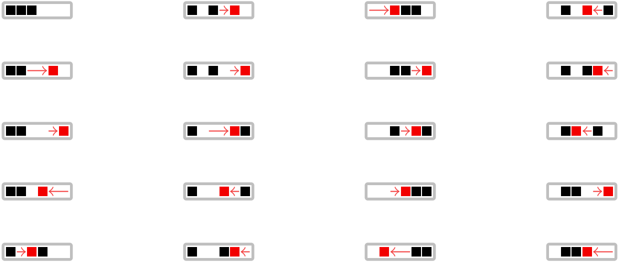
$$\mathrm{rk}(\sigma, S) = 3 \times 120 + 24 + 6 + 2 = 392$$

- We rank patterns based on their inversions:

$$\mathrm{rk}(\sigma, S) = \sum_{i,j \in S,\, i<j,\, \sigma_i > \sigma_j} |S_{>i}|!$$

- Computing the rank of every pattern of every permutation (up to symmetries) can be done in time $\frac{n!}{8} \times \binom{n}{k} \times \binom{k}{2}$.

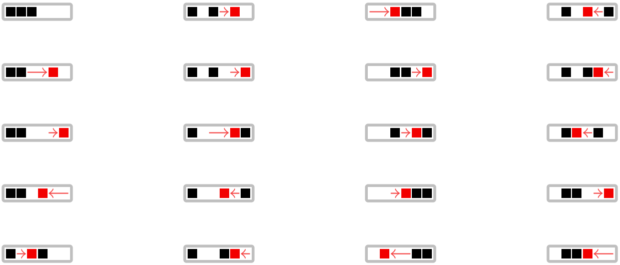
Iterating over subsets: a combinatorial Gray code



Theorem (Chase, 1976)

There exists an enumeration of $\mathfrak{P}_k[n]$ moving one point at a time, without crossing other points.

Iterating over subsets: a combinatorial Gray code

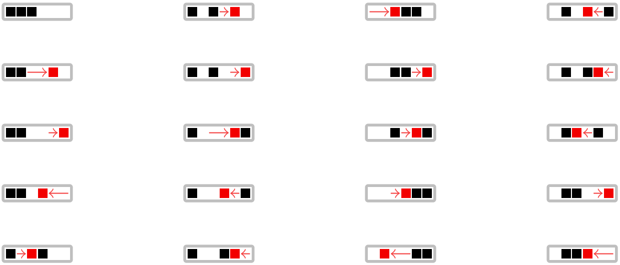


Theorem (Chase, 1976)

There exists an enumeration of $\mathfrak{P}_k[n]$ moving one point at a time, without crossing other points.

- At each step, going from $\mathrm{rk}(\sigma, S)$ to $\mathrm{rk}(\sigma, S')$ takes time k.

Iterating over subsets: a combinatorial Gray code

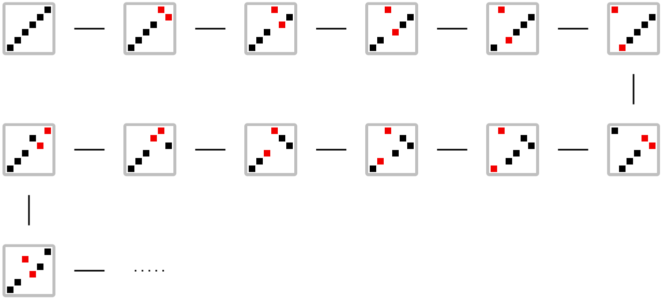


Theorem (Chase, 1976)

There exists an enumeration of $\mathfrak{P}_k[n]$ moving one point at a time, without crossing other points.

- At each step, going from $\mathrm{rk}(\sigma, S)$ to $\mathrm{rk}(\sigma, S')$ takes time k.
- This improves the complexity to $\frac{n!}{8} \times \binom{n}{k} \times k$.

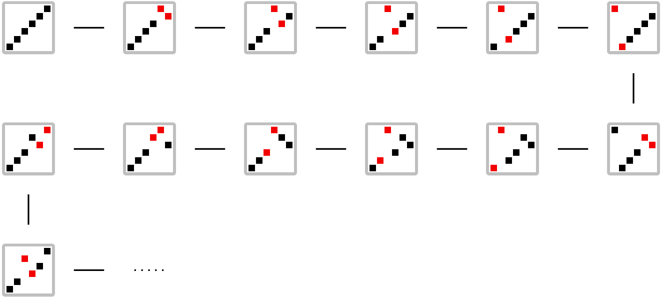
Iterating over permutations: another Gray code



Theorem (Johnson, 1963; Trotter, 1962)

There exists an enumeration of $\mathfrak{S}_n$ doing only elementary transpositions.

Iterating over permutations: another Gray code

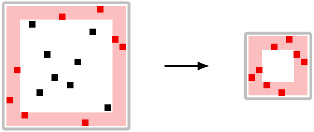


Theorem (Johnson, 1963; Trotter, 1962)

There exists an enumeration of $\mathfrak{S}_n$ doing only elementary transpositions.

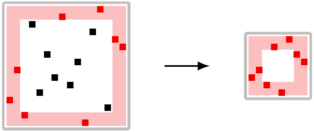
- Problem: how to iterate on permutations up to symmetries?

Iterating over permutations, exploiting symmetries

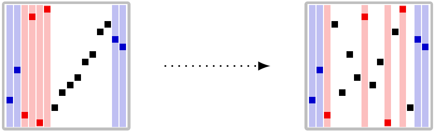


- We divide permutations into classes based on their *m*-border pattern.
- We discard symmetrical classes ($m = 2$: $\approx 85\%$ of permutations).

Iterating over permutations, exploiting symmetries

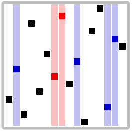


- We divide permutations into classes based on their *m*-border pattern.
- We discard symmetrical classes ($m = 2$: $\approx 85\%$ of permutations).



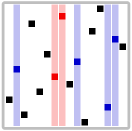
- Classes are divided into batches by fixing entries to the left and right.
- Each batch has $\frac{(n-2m)!}{(2m)!}$ permutations and a Gray code.

Algorithm 1 (small patterns)



- Swapping σ_i and σ_{i+1} only affects patterns containing both.
- In Chase order, computing $\mathrm{rk}(\sigma, S)$ and $\mathrm{rk}(\sigma', S)$ takes k operations.

Algorithm 1 (small patterns)



- Swapping σ_i and σ_{i+1} only affects patterns containing both.
- In Chase order, computing $\mathrm{rk}(\sigma, S)$ and $\mathrm{rk}(\sigma', S)$ takes k operations.

Algorithm 1

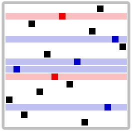
Remember: $c_r = \#\{S \mid \mathrm{rk}(\sigma, S) = r\}$ for $0 \leq r < k!$.

Step $\sigma \xrightarrow{e_i} \sigma'$: For all $S \supset \{i, i+1\}$ in Chase order:

- compute $r = \mathrm{rk}(\sigma, S)$ and $r' = \mathrm{rk}(\sigma', S)$;
- decrement c_r and increment $c_{r'}$.

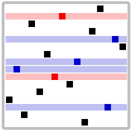
Complexity: $\frac{n!}{8} \times \binom{n-2}{k-2} \times k$ with $k!$ space.

Algorithm 2 (large patterns)



- Swapping σ_i and σ_{i+1} changes $\mathrm{rk}(\sigma^{-1}, S)$ only if $\{\sigma_i, \sigma_{i+1}\} \subset S$ and only by the contribution of the inversion (σ_i, σ_{i+1}).

Algorithm 2 (large patterns)



- Swapping σ_i and σ_{i+1} changes $\mathrm{rk}(\sigma^{-1}, S)$ only if $\{\sigma_i, \sigma_{i+1}\} \subset S$ and only by the contribution of the inversion (σ_i, σ_{i+1}).

Algorithm 2

Remember: $r_S = \mathrm{rk}(\sigma^{-1}, S)$ for $S \in \mathfrak{P}_k[n]$.

Step $\sigma \xrightarrow{e_i} \sigma'$: Initialize a hash table, then for all $S \in \mathfrak{P}_k[n]$, do:

- if $\{\sigma_i, \sigma_{i+1}\} \subset S$, $\quad r_S \leftarrow \begin{cases} r_S + |S_{>\sigma_i}|! & \text{if } \sigma_i < \sigma_{i+1}, \\ r_S - |S_{>\sigma_{i+1}}|! & \text{if } \sigma_i > \sigma_{i+1}; \end{cases}$
- add r_S to the table.

Complexity: $\frac{n!}{8} \times \binom{n}{k}$ with $\binom{n}{k}$ space.

Threads and memory on a GPU

- Threads on a GPU are organized in warps (32 threads per warp).
- Warps are (usually) always synchronized and threads can read each other's registers.

- Warps are organized in blocks (1–32 warps per block).
- Blocks may be synchronized and have access to shared memory.
- Limits: 1024 resident threads, 65536 32-bit registers and 64 kB of shared memory per multiprocessor (46 MPs per GPU).

- Threads in different blocks cannot synchronize (except for atomics).
- They have access to the global memory of the GPU (8 GB) through different caches.

CUDA programming

```
__global__ void search(perm_t *batches) {
    perm_t p = batches[blockIdx.x];
    /*...*/
}
```

```
int main() {
    /*...*/
    search <<< num_batches, 512 >>> (batches);
    /*...*/
}
```

- The above CPU code launches the kernel `search()` with `num_batches` blocks of 512 threads each.
- Threads have access to their block number (`blockIdx.x`) and thread number within their block (`threadIdx.x`).
- An API exists for memory allocation, copy, config, etc.

Algorithm 1: implementation

Algorithm 1

Remember: $c_r = \#\{S \mid \mathrm{rk}(\sigma, S) = r\}$ for $0 \le r < k!$.

Step $\sigma \xrightarrow{e_i} \sigma'$: For all $S \supset \{i, i+1\}$ in Chase order:

- compute $r = \mathrm{rk}(\sigma, S)$ and $r' = \mathrm{rk}(\sigma', S)$;
- decrement c_r and increment $c_{r'}$.

Complexity: $\frac{n!}{8} \times \binom{n-2}{k-2} \times k$ with $k!$ space.

- We need $2k!$ bytes of shared memory per permutation for (c_r).
- If $k \le 6$, we fit 32 permutations per MP (1 warp/permutation).
- If $k = 7$, we fit 6 permutations per MP (5 warps/permutation).
- We fit 2 permutations per block (64 or 320 threads/block).
- The Chase orders are precomputed and stored in global memory.

Algorithm 2: implementation

Algorithm 2

Remember: $r_S = \mathrm{rk}(\sigma^{-1}, S)$ for $S \in \mathfrak{P}_k[n]$.

Step $\sigma \xrightarrow{e_i} \sigma'$: Initialize a hash table, then for all $S \in \mathfrak{P}_k[n]$, do:

- if $\{\sigma_i, \sigma_{i+1}\} \subset S$, $\quad r_S \leftarrow \begin{cases} r_S + |S_{>\sigma_i}|! & \text{if } \sigma_i < \sigma_{i+1}, \\ r_S - |S_{>\sigma_{i+1}}|! & \text{if } \sigma_i > \sigma_{i+1}; \end{cases}$
- add r_S to the table.

Complexity: $\frac{n!}{8} \times \binom{n}{k}$ with $\binom{n}{k}$ space.

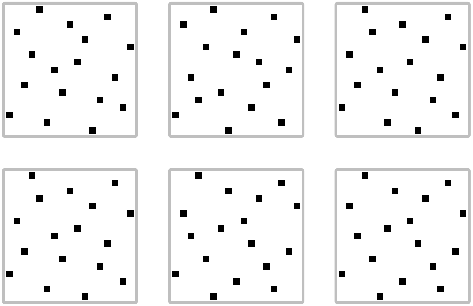
- When $\binom{n}{k}$ is large, we use 1024 threads per block.
- We store S and r_S in registers (works well for $\binom{n}{k} \lesssim 20000$).
- The hash table is in shared memory.
- Global memory is only needed for writing optimal permutations.

Hash table implementation

```
__shared__ unsigned int table[TABLE_SIZE];

__device__ void table_zero() {
    for(unsigned int i = threadIdx.x; i < TABLE_SIZE; i += blockDim.x)
        table[i] = 0;
}

__device__ unsigned int hash(unsigned int key) { /*...*/ }

// returns 1 if key was not in table, 0 otherwise
__device__ int table_add(unsigned int key) {
    unsigned int i = hash(key);
    while(1) {
        unsigned int t = atomicCAS(table + i, 0, key);
        if(t == 0 || t == key) return t == 0;
        i = (i+1) % TABLE_SIZE;
    }
}
```

- `t = atomicCAS(p, x, y);` $\Leftrightarrow$ `{ t = *p; if(t == x) *p = y; }`
- Maximum size of the table: 16384 entries (best when $\lesssim$ 50% full).

Perspectives



- What are the permutations with the most patterns of all sizes? (currently found for $n \leq 15$ by adapting Algorithm 2)
- What to do when there are $\gg 8000$ different patterns?
- Can we find necessary conditions for optimal permutations and discard batches *a priori*?