An Efficient Skip-Search Approach to the Order-Preserving Pattern Matching Problem

Domenico Cantone, Simone Faro and M. Oğuzhan Külekci

Department of Mathematics and Computer Science, University of Catania (Italy)
ERLAB Software Co. ITU ARI Teknokent, Istanbul (Turkey)

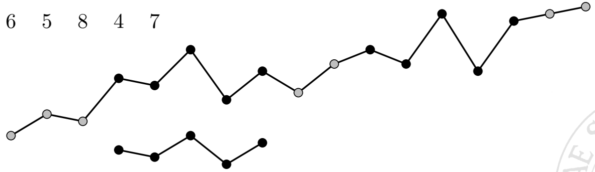
The Prague Stringology Conference 2015
Prague 24-26 August, 2015

The Order-Preserving Pattern Matching Problem

Given a pattern *x* and a text *y*, both over a common ordered alphabet, the *order-preserving pattern matching* problem consists in finding all substrings of the text with the same relative order as the pattern.

$$y = 8 \quad 11 \quad 10 \quad \underline{16} \quad \underline{15} \quad \underline{20} \quad \underline{13} \quad \underline{17} \quad 14 \quad 18 \quad \underline{20} \quad \underline{18} \quad \underline{25} \quad \underline{17} \quad \underline{24} \quad 25 \quad 26$$

$$x = 6 \quad 5 \quad 8 \quad 4 \quad 7$$



The Order-Preserving Pattern Matching Problem

Given a pattern x and a text y, both over a common ordered alphabet, the *order-preserving pattern matching* problem consists in finding all substrings of the text with the same relative order as the pattern.

In this paper we present a new efficient approach to this problem :

- it is inspired to the well-known Skip Search algorithm;
- It makes use of efficient SIMD SSE instructions;
- it is up to twice as faster than previous solutions.

The State of Art

2013 The first solution was presented by Kubica *et al.*. They proposed a $\mathcal{O}(n + m \log m)$ solution over generic ordered alphabets based on the KMP algorithm and a $\mathcal{O}(n + m)$ solution in the case of integer alphabets.

The State of Art

2013 The first solution was presented by Kubica *et al.*. They proposed a $\mathcal{O}(n + m \log m)$ solution over generic ordered alphabets based on the KMP algorithm and a $\mathcal{O}(n + m)$ solution in the case of integer alphabets.

2013 Kim *et al.* presented a similar solution running in $\mathcal{O}(n + m \log m)$ time, still based on the KMP approach.

The State of Art

2013 The first solution was presented by Kubica *et al.*. They proposed a $\mathcal{O}(n + m \log m)$ solution over generic ordered alphabets based on the KMP algorithm and a $\mathcal{O}(n + m)$ solution in the case of integer alphabets.

2013 Kim *et al.* presented a similar solution running in $\mathcal{O}(n + m \log m)$ time, still based on the KMP approach.

2014 Cho *et al.* showed that the Boyer-Moore approach can be applied also to the order-preserving variant of the pattern matching problem.

The State of Art

2013 The first solution was presented by Kubica *et al.*. They proposed a $\mathcal{O}(n + m \log m)$ solution over generic ordered alphabets based on the KMP algorithm and a $\mathcal{O}(n + m)$ solution in the case of integer alphabets.

2013 Kim *et al.* presented a similar solution running in $\mathcal{O}(n + m \log m)$ time, still based on the KMP approach.

2014 Cho *et al.* showed that the Boyer-Moore approach can be applied also to the order-preserving variant of the pattern matching problem.

2014 Chhabra and Tarhio presented a more practical solution based on approximate string matching techniques.

The State of Art

2013 The first solution was presented by Kubica *et al.*. They proposed a $\mathcal{O}(n + m \log m)$ solution over generic ordered alphabets based on the KMP algorithm and a $\mathcal{O}(n + m)$ solution in the case of integer alphabets.

2013 Kim *et al.* presented a similar solution running in $\mathcal{O}(n + m \log m)$ time, still based on the KMP approach.

2014 Cho *et al.* showed that the Boyer-Moore approach can be applied also to the order-preserving variant of the pattern matching problem.

2014 Chhabra and Tarhio presented a more practical solution based on approximate string matching techniques.

2015 Our Solution

Our Solution

In this paper we present a new algorithm for the OPPM problem:

idea. *The algorithm is based on the well-known Skip Search algorithm for the exact string matching problem, which consists in processing separately chunks of the text for any occurrence of the pattern.*

Our Solution

In this paper we present a new algorithm for the OPPM problem:

- idea. *The algorithm is based on the well-known Skip Search algorithm for the exact string matching problem, which consists in processing separately chunks of the text for any occurrence of the pattern.*
- tech. *We use efficient SIMD SSE instructions for computing the fingerprints of the pattern substrings.*

Our Solution

In this paper we present a new algorithm for the OPPM problem:

- idea. *The algorithm is based on the well-known Skip Search algorithm for the exact string matching problem, which consists in processing separately chunks of the text for any occurrence of the pattern.*
- tech. *We use efficient SIMD SSE instructions for computing the fingerprints of the pattern substrings.*
- results. *Experimental results show that our proposed approach leads to algorithmic variants that are up to twice as faster than previous solutions present in the literature.*

The Skip-Search Algorithm

It is an elegant and efficient solution to the exact pattern matching problem, presented in 1998, subsequently adapted to many other problems and variants of the exact pattern matching problem.

The Skip-Search Algorithm

It is an elegant and efficient solution to the exact pattern matching problem, presented in 1998, subsequently adapted to many other problems and variants of the exact pattern matching problem.

How Does It Works?

Let x and y be a pattern and a text of length m and n, respectively, over a common alphabet Σ of size σ. For each character c of the alphabet, the Skip Search algorithm collects in a bucket $B[c]$ all the positions of that character in the pattern x, so that for each $c \in \Sigma$ we have:

$$B[c] = \{i \ : \ 0 \leq i \leq m-1 \text{ and } x[i] = c\}.$$

The Skip-Search Algorithm

It is an elegant and efficient solution to the exact pattern matching problem, presented in 1998, subsequently adapted to many other problems and variants of the exact pattern matching problem.

How Does It Works?

Let x and y be a pattern and a text of length m and n, respectively, over a common alphabet Σ of size σ. For each character c of the alphabet, the Skip Search algorithm collects in a bucket $B[c]$ all the positions of that character in the pattern x, so that for each $c \in \Sigma$ we have:

$$x = \langle acbccbabbac \rangle, B[a] = \{0, 6, 9\}, B[b] = \{2, 5, 7, 8\}.$$

The Search Phase of the Skip-Search Algorithm

- The algorithm examines all the characters $y[j]$ in the text at positions $j = km - 1$, for $k = 1, 2, \ldots, \lfloor n/m \rfloor$.
- For each character $y[j]$, the bucket $B[y[j]]$ allows one to compute the possible positions h at which the pattern could occur.
- a character-by-character comparison between x and the subsequence $y[h \mathinner{..} h + m - 1]$ is performed.

The Search Phase of the Skip-Search Algorithm

- The algorithm examines all the characters $y[j]$ in the text at positions $j = km - 1$, for $k = 1, 2, \ldots, \lfloor n/m \rfloor$.
- For each character $y[j]$, the bucket $B[y[j]]$ allows one to compute the possible positions h at which the pattern could occur.
- a character-by-character comparison between x and the subsequence $y[h \,..\, h + m - 1]$ is performed.

Example

$$
\begin{array}{l}
x = \langle acbccbabbac \rangle \\
y = \langle acbaccabcb\underline{a}bcbabbccbabccbaa \rangle \\
\quad\quad \langle acbccbabb\underline{a}c \rangle \\
\quad\quad\quad\quad \langle acbccb\underline{a}bbac \rangle \\
\quad\quad\quad\quad\quad\quad\quad\quad \langle \underline{a}cbccbabbac \rangle
\end{array}
$$

The Model

- Word-Ram Model of Computation
- Packed String Matching
  - w-bit register
  - alphabet of size σ
  - α is the packing factor, with $\alpha = \lfloor w / \log \sigma \rfloor$

The instruction wsrv (word size rank vector)

$$r[0\,..\,\alpha-1] = \mathbf{wsrv}(B[0\,..\,\alpha-1], i), \text{ where}$$
$$r[j] = 1 \text{ iff } B[i] \geq B[j], \text{ and } r[j] = 0 \text{ otherwise.}$$

Example

In the following example suppose $\alpha = 8$
let $B = \langle 2, 7, 4, 1, 9, 8, 10, 3 \rangle$ *and let* $i = 1$
if $r = \mathbf{wsrv}(B, i)$ *then* $r = \langle 1, 1, 1, 1, 0, 0, 0, 1 \rangle$

The instruction wsrv (word size rank vector)

$r[0\mathinner{\ldotp\ldotp}\alpha-1] = \mathbf{wsrv}(B[0\mathinner{\ldotp\ldotp}\alpha-1], i)$, where
$r[j] = 1$ iff $B[i] \geq B[j]$, and $r[j] = 0$ otherwise.

Observe!

let $B_1 = \langle 2, 7, 4, 1, 9, 8, 10, 3\rangle$ *and* $B_2 = \langle 6, 11, 8, 5, 13, 12, 14, 7\rangle$
$r_1 = \mathbf{wsrv}(B_1, 1) = \langle 1, 1, 1, 1, 0, 0, 0, 1\rangle$
$r_2 = \mathbf{wsrv}(B_2, 1) = \langle 1, 1, 1, 1, 0, 0, 0, 1\rangle$

The instruction wsrv (word size rank vector)

$$r[0\mathinner{.\,.}\alpha-1] = \mathbf{wsrv}(B[0\mathinner{.\,.}\alpha-1], i), \text{ where}$$
$$r[j] = 1 \text{ iff } B[i] \geq B[j], \text{ and } r[j] = 0 \text{ otherwise.}$$

Simulation

The wsrv$(B[0\mathinner{.\,.}\alpha-1], i)$ specialized instruction can be emulated in constant time by the following sequence of specialized SIMD instructions:

$wsrv(B, i)$

$D \leftarrow$ *_mm_set1_epi8*$(B[i])$

$C \leftarrow$ *_mm_cmpgt_epi8*(B, D)

$r \leftarrow$ *_mm_movemask_epi8*(C)

return r

The instruction wsrp (word size relative position)

$r[0\,..\,\alpha-1] = \mathbf{wsrp}(B[0\,..\,\alpha-1])$, where
$r[j] = 1$ iff $B[j] \geq B[j+1]$, and $r[j] = 0$ otherwise (we put $r[\alpha-1] = 0$).

Example

In the following example suppose $\alpha = 8$
let $B = \langle 2, 7, 4, 1, 9, 8, 10, 3 \rangle$ *and let* $i = 1$
if $r = \mathbf{wsrp}(B)$ *then* $r = \langle 0, 1, 1, 0, 1, 0, 1, 0 \rangle$

The instruction wsrp (word size relative position)

$r[0\,..\,\alpha-1] = \textbf{wsrp}(B[0\,..\,\alpha-1])$, where
$r[j] = 1$ iff $B[j] \geq B[j+1]$, and $r[j] = 0$ otherwise (we put $r[\alpha-1] = 0$).

Observe!

let $B_1 = \langle 2, 7, 4, 1, 9, 8, 10, 3 \rangle$ *and* $B_2 = \langle 6, 11, 8, 5, 13, 12, 14, 7 \rangle$
$r_1 = \textbf{wsrp}(B_1) = \langle 0, 1, 1, 0, 1, 0, 1, 0 \rangle$
$r_2 = \textbf{wsrp}(B_2) = \langle 0, 1, 1, 0, 1, 0, 1, 0 \rangle$

The instruction wsrp (word size relative position)

$r[0 \mathinner{..} \alpha-1] = \textbf{wsrp}(B[0 \mathinner{..} \alpha-1])$, where
$r[j] = 1$ iff $B[j] \geq B[j+1]$, and $r[j] = 0$ otherwise (we put $r[\alpha-1] = 0$).

Simulation
The wsrp(B) specialized instruction can be emulated in constant time by the following sequence of specialized SIMD instructions

wsrp(B)

$D \leftarrow$ *_mm_slli_si128*$(B, 1)$
$C \leftarrow$ *_mm_cmpgt_epi8*(B, D)
$r \leftarrow$ *_mm_movemask_epi8*(C)
return r

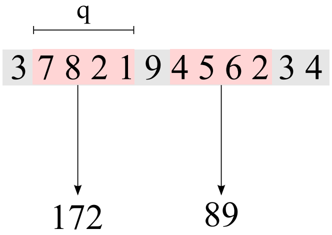
The Preprocessing Phase

- We index the subsequences of the pattern (of length q) in order to locate them during the searching phase.
- Each numeric sequence of length q is converted into a numeric value, called *fingerprint*, which is used to index the substring.
- A fingerprint value ranges in the interval $\{0 .. \tau - 1\}$, for a given bound τ. The value τ is set to 2^{16}, so that a fingerprint can fit into a single 16-bit register.

3 7 8 2 1 9 4 5 6 2 3 4

The Preprocessing Phase

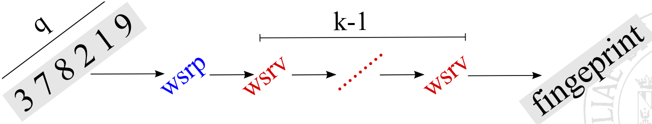
- We index the subsequences of the pattern (of length q) in order to locate them during the searching phase.
- Each numeric sequence of length q is converted into a numeric value, called *fingerprint*, which is used to index the substring.
- A fingerprint value ranges in the interval $\{0 \mathinner{..} \tau - 1\}$, for a given bound τ. The value τ is set to 2^{16}, so that a fingerprint can fit into a single 16-bit register.



How to Compute the Fingerprint Values

- x is a sequence of length m inserted in the register B
- q is the size of the q-gram
- i is the initial position of the q-gram
- k is the number of instructions we use for computing the fingerprint

$$v = \text{wsrp}(B) \times 2^{k-1} + \sum_{j=0}^{k-2} \left(\text{wsrv}(B, \alpha - q + j) \times 2^{k-2-j}\right) .$$



$\textsc{fng}(x, i, q, k)$

$B \leftarrow 0^{\alpha-q}.x[i\,..\,i+q-1]$
$v \leftarrow \mathsf{wsrp}(B)$
for $j \leftarrow 0$ to $k-2$ do
  $v \leftarrow (v \ll 1) + \mathsf{wsrv}(B, \alpha - q + j)$
return v

$\textsc{Example}$ ($q = 5$, $k = 3$, and $\alpha = 8$)

$$x[i\,..\,i+q-1] = \langle 3, 6, 2, 4, 7 \rangle$$
$$B = [0,0,0,3,6,2,4,7]$$
$$\mathsf{wsrp}(B) = [0,0,0,1,0,1,1,0] = 22_{10}$$
$$\mathsf{wsrv}(B,3) = [0,0,0,1,1,0,1,1] = 27_{10}$$
$$\mathsf{wsrv}(B,4) = [0,0,0,0,1,0,0,1] = 9_{10}$$
$$v = 22 \times 2^2 + 27 \times 2^1 + 9 = 151_{10}$$

The Preprocessing Phase

The preprocessing phase of the SKSOP algorithm consists in compiling the fingerprints of all possible substrings of length q contained in the pattern x.

$$F[v] = \Big\{ i \mid 0 \leq i < m - q \text{ and } \textsc{fng}(x, i, q, k) = v \Big\}.$$

The Searching Phase

- The main loop investigates the blocks of the text y in steps of $(m - q + 1)$ blocks.
- If the fingerprint v computed on $y[j \,..\, j + q - 1]$ points to a nonempty bucket of the table F, then the positions listed in $F[v]$ are verified accordingly.
- While looking for occurrences on $y[j \,..\, j + q - 1]$, if $F[v]$ contains the value i, this indicates the pattern x may potentially begin at position $(j - i)$ of the text.
- In that case, a matching test is to be performed between x and $y[j - i \,..\, j - i + m - 1]$ via a character-by-character inspection.

$\textsc{Preprocessing}(x, q, m, k)$
for $v \leftarrow 0$ to $2^{\alpha} - 1$ do
  $F[v] \leftarrow \emptyset$
for $i \leftarrow 0$ to $m - q$ do
  $v \leftarrow \textsc{fng}(x, i, q, k)$
  $F[v] \leftarrow F[v] \cup \{(i + q - 1)\}$
return F

$\textsc{SkSop}(x, r, y, n, q, k)$
$F \leftarrow$Preprocessing(x, q, m, k)
for $j \leftarrow m - 1$ to n step $m - q + 1$ do
  $v \leftarrow \textsc{fng}(y, j, q, k)$
  for each $i \in F[v]$ do
    $z \leftarrow y[j - i \,..\, j - i + m - 1]$
    if $\textsc{Order-Isomorphic}(rk_x^{-1}, eq_x, z)$
      then output (j)

Space and Time Analysis

- The total number of filtering operations is exactly $n/(m-q)$.
- At each attempt, the maximum number of verification requests is $(m-q)$.
- The verification cost for a pattern x of length m is assumed to be $\mathcal{O}(m)$, with the brute-force checking approach.
- Hence, in the worst case the time complexity of the verification is $\mathcal{O}(m(m-q))$, which happens when all alignments in x must be verified at any possible beginning position.
- Hence, the best case complexity is $\mathcal{O}(n/(m-q))$, while the worst case complexity is $\mathcal{O}(nm)$.

Experimental Settings: Algorithms

In our experimental evaluations, we used the SBNDM2 algorithm to test the filter approach by Chhabra and Tarhio. In our dataset we use the following short names to identify the algorithms that we have tested:

- FCT: the SBNDM2 algorithm based on the filter approach by Chhabra and Tarhio;
- SKSOP(k, q): our SKIP SEARCH-based algorithm which combines k different fingerprint values on subsequences of length q.

Experimental Settings: Data Sets

We tested our algorithm on the following set of small integer sequences. Each text consists in a sequence of 1 million elements.

- Rand-δ: a sequence of random integer values ranging around a fixed mean μ with a variability of δ and a uniform distribution, i.e. each value is uniformly distributed in the range $\{\mu - \delta \mathinner{..} \mu + \delta\}$;
- Periodic-ρ: a sequence of random integer values uniformly ranging around a cyclic function with a period of ρ elements.

δ	m	SkSop(1, q)	SkSop(2, q)	SkSop(3, q)	SkSop(4, q)	SkSop(5, q)
5	8	52.64 (8)	3.87 (8)	1.20 (8)	<u>0.25</u> (8)	0.43 (8)
	12	46.22 (8)	4.27 (8)	1.02 (8)	<u>0.25</u> (8)	0.41 (8)
	16	45.65 (8)	4.01 (8)	1.06 (8)	<u>0.24</u> (8)	0.41 (8)
	20	48.13 (8)	4.18 (8)	1.09 (8)	<u>0.24</u> (8)	0.42 (8)
	24	45.02 (8)	4.13 (8)	1.05 (8)	<u>0.24</u> (8)	0.42 (8)
	28	44.92 (8)	4.05 (8)	1.03 (8)	<u>0.24</u> (8)	0.41 (8)
	32	46.99 (8)	4.23 (8)	1.04 (8)	<u>0.23</u> (8)	0.44 (8)
20	8	37.78 (8)	3.96 (8)	1.02 (8)	<u>0.23</u> (8)	0.51 (8)
	12	40.65 (8)	4.17 (8)	1.04 (8)	<u>0.25</u> (8)	0.52 (8)
	16	39.78 (8)	4.65 (8)	1.00 (8)	<u>0.25</u> (8)	0.52 (8)
	20	39.05 (8)	4.12 (8)	1.02 (8)	<u>0.24</u> (8)	0.49 (8)
	24	39.24 (8)	4.35 (8)	1.02 (8)	<u>0.25</u> (8)	0.50 (8)
	28	40.15 (8)	4.34 (8)	1.00 (8)	<u>0.24</u> (8)	0.49 (8)
	32	40.00 (8)	4.39 (8)	1.01 (8)	<u>0.25</u> (8)	0.51 (8)
40	8	42.34 (8)	4.37 (8)	1.03 (8)	<u>0.27</u> (8)	0.54 (8)
	12	35.64 (8)	4.50 (8)	0.99 (8)	<u>0.25</u> (8)	0.49 (8)
	16	41.08 (8)	4.40 (8)	1.01 (8)	<u>0.26</u> (8)	0.54 (8)
	20	40.71 (8)	4.29 (8)	1.05 (8)	<u>0.26</u> (8)	0.54 (8)
	24	37.77 (8)	4.33 (8)	0.96 (8)	<u>0.25</u> (8)	0.52 (8)
	28	39.98 (8)	4.51 (8)	1.02 (8)	<u>0.25</u> (8)	0.53 (8)
	32	38.26 (8)	4.46 (8)	0.99 (8)	<u>0.26</u> (8)	0.54 (8)

Tabella: Average number of verifications performed every 2^{10} characters, computed on a Rand-δ small integer sequence, with $\delta = 5, 20$, and 40.

ρ	m	SkSop(1, q)	SkSop(2, q)	SkSop(3, q)	SkSop(4, q)	SkSop(5, q)
8	8	92.22 (8)	37.40 (8)	14.22 (8)	8.01 (8)	8.83 (8)
	12	98.48 (8)	35.46 (8)	14.72 (8)	8.27 (8)	10.38 (8)
	16	98.27 (8)	36.46 (8)	15.71 (8)	8.77 (8)	10.46 (8)
	20	96.95 (8)	35.91 (8)	15.14 (8)	8.47 (8)	10.12 (8)
	24	96.88 (8)	36.06 (8)	14.87 (8)	8.34 (8)	10.18 (8)
	28	97.63 (8)	35.79 (8)	14.60 (8)	7.94 (8)	9.67 (8)
	32	97.65 (8)	35.93 (8)	15.09 (8)	8.31 (8)	10.23 (8)
16	8	173.85 (8)	40.23 (8)	5.19 (8)	3.74 (8)	7.03 (8)
	12	179.84 (8)	46.64 (8)	5.35 (8)	4.25 (8)	7.62 (8)
	16	179.20 (8)	46.94 (8)	5.59 (8)	4.35 (8)	7.57 (8)
	20	176.24 (8)	45.61 (8)	5.40 (8)	4.20 (8)	7.07 (8)
	24	181.67 (8)	46.50 (8)	5.53 (8)	4.24 (8)	7.31 (8)
	28	176.67 (8)	46.27 (8)	5.47 (8)	4.18 (8)	7.09 (8)
	32	179.96 (8)	46.12 (8)	5.55 (8)	4.34 (8)	7.52 (8)
32	8	125.55 (8)	35.52 (8)	3.23 (8)	2.26 (8)	3.96 (8)
	12	134.48 (8)	35.41 (8)	3.19 (8)	2.13 (8)	3.84 (8)
	16	136.69 (8)	39.27 (8)	3.34 (8)	2.31 (8)	4.07 (8)
	20	140.14 (8)	40.58 (8)	3.51 (8)	2.33 (8)	3.99 (8)
	24	138.36 (8)	39.70 (8)	3.52 (8)	2.39 (8)	4.15 (8)
	28	139.04 (8)	37.90 (8)	3.44 (8)	2.35 (8)	4.05 (8)
	32	136.39 (8)	39.09 (8)	3.44 (8)	2.33 (8)	4.06 (8)

Tabella: Average number of verifications performed every 2^{10} characters, computed on a Periodic-ρ small integer sequence, with $\rho = 8, 16$, and 32.

δ	m	FCT	SKSOP(1, q)	SKSOP(2, q)	SKSOP(3, q)	SKSOP(4, q)	SKSOP(5, q)
	8	42.32	0.81 (5)	1.14 (4)	1.22 (4)	1.27 (4)	1.27 (4)
	12	27.09	0.80 (7)	1.21 (5)	1.35 (5)	1.37 (5)	1.34 (5)
	16	20.38	0.83 (8)	1.33 (6)	1.44 (5)	1.52 (5)	1.50 (5)
5	20	16.59	0.88 (8)	1.39 (7)	1.54 (6)	1.58 (5)	1.56 (6)
	24	13.56	0.89 (8)	1.44 (7)	1.60 (6)	1.61 (6)	1.63 (6)
	28	11.50	0.85 (8)	1.47 (7)	1.56 (7)	1.59 (5)	1.62 (6)
	32	9.97	0.81 (8)	1.47 (7)	1.57 (7)	1.59 (6)	1.60 (6)
	8	42.13	0.81 (4)	1.12 (4)	1.22 (4)	1.19 (4)	1.14 (4)
	12	27.41	0.84 (6)	1.24 (5)	1.40 (5)	1.40 (5)	1.35 (5)
	16	19.78	0.85 (7)	1.28 (6)	1.43 (6)	1.46 (5)	1.40 (5)
20	20	15.73	0.90 (8)	1.33 (7)	1.49 (6)	1.51 (5)	1.50 (6)
	24	13.24	0.89 (8)	1.40 (7)	1.51 (6)	1.55 (6)	1.55 (6)
	28	11.37	0.86 (8)	1.45 (7)	1.57 (6)	1.57 (6)	1.58 (6)
	32	9.89	0.85 (8)	1.42 (7)	1.58 (7)	1.56 (6)	1.54 (7)
	8	41.32	0.81 (4)	1.11 (4)	1.19 (4)	1.16 (4)	1.11 (4)
	12	27.36	0.83 (6)	1.22 (5)	1.38 (5)	1.39 (5)	1.34 (5)
	16	19.78	0.84 (7)	1.27 (6)	1.42 (6)	1.43 (5)	1.40 (6)
40	20	16.21	0.90 (8)	1.34 (7)	1.51 (6)	1.52 (6)	1.52 (6)
	24	13.26	0.90 (8)	1.40 (7)	1.51 (7)	1.54 (6)	1.57 (6)
	28	11.38	0.86 (8)	1.43 (7)	1.56 (7)	1.56 (6)	1.57 (6)
	32	9.93	0.84 (8)	1.43 (8)	1.56 (7)	1.58 (6)	1.58 (7)

Tabella: Running times on a RAND-δ small integer sequence, with $\delta = 5, 20$ and 40. Running times (in milliseconds) are reported for the FCT algorithm, while speed-up values are reported for the SKSOP(k, q) algorithms.

ρ	m	FCT	SKSOP(1, q)	SKSOP(2, q)	SKSOP(3, q)	SKSOP(4, q)	SKSOP(5, q)
8	8	39.90	0.79 (3)	0.87 (4)	0.89 (4)	0.87 (4)	0.87 (4)
	12	32.94	0.87 (5)	1.09 (6)	1.19 (6)	1.24 (6)	1.17 (6)
	16	27.21	0.90 (7)	1.24 (7)	1.44 (7)	1.55 (7)	1.46 (7)
	20	21.47	0.90 (8)	1.21 (7)	1.44 (7)	1.60 (7)	1.50 (7)
	24	19.29	0.94 (8)	1.27 (7)	1.59 (8)	1.77 (7)	1.68 (8)
	28	16.90	0.91 (8)	1.28 (7)	1.65 (8)	1.81 (7)	1.77 (8)
	32	15.58	0.89 (8)	1.28 (7)	1.68 (8)	1.87 (8)	1.83 (8)
16	8	37.40	0.68 (4)	0.83 (4)	0.93 (4)	0.93 (4)	0.90 (4)
	12	25.03	0.60 (5)	0.79 (4)	1.03 (5)	1.04 (5)	0.99 (5)
	16	18.63	0.60 (6)	0.80 (7)	1.15 (6)	1.15 (6)	1.10 (6)
	20	15.22	0.53 (8)	0.81 (8)	1.22 (7)	1.19 (7)	1.15 (7)
	24	12.75	0.50 (7)	0.78 (8)	1.26 (7)	1.24 (7)	1.19 (7)
	28	10.52	0.45 (7)	0.73 (8)	1.21 (7)	1.20 (7)	1.14 (7)
	32	10.01	0.43 (8)	0.78 (8)	1.31 (8)	1.29 (7)	1.23 (8)
32	8	38.82	0.76 (4)	1.00 (4)	1.11 (4)	1.09 (4)	1.05 (4)
	12	24.86	0.65 (6)	0.91 (4)	1.16 (5)	1.18 (5)	1.15 (5)
	16	18.85	0.61 (6)	0.89 (5)	1.24 (6)	1.27 (5)	1.24 (6)
	20	15.02	0.58 (6)	0.86 (6)	1.31 (6)	1.34 (6)	1.30 (6)
	24	12.32	0.52 (7)	0.83 (7)	1.31 (7)	1.32 (6)	1.28 (6)
	28	10.89	0.50 (8)	0.85 (8)	1.38 (7)	1.38 (6)	1.34 (6)
	32	9.50	0.48 (8)	0.81 (8)	1.37 (7)	1.38 (6)	1.34 (7)

Tabella: Running times on a PERIODIC-δ integer sequence, with $\delta = 5, 20$, and 40. Running times (in milliseconds) are reported for the FCT algorithm, while speed-up values are reported for the SKSOP(k, q) algorithms.

Thank You!