Received: 29 December 2018 | Revised: 30 May 2019 | Accepted: 5 June 2019
DOI: 10.1002/cpe.5448



WILEY

SPECIAL ISSUE PAPER

# JPEG steganography with particle swarm optimization accelerated by AVX

Vaclav Snasel[1] | Pavel Kromer[1] | Jakub Safarik[2] | Jan Platos[1]

[1]Department of Computer Science, FEECS, VSB-Technical University of Ostrava, Ostrava, Czech Republic
[2]Laboratory of Big Data Analysis, IT4Innovations, VSB-Technical University of Ostrava, Ostrava, Czech Republic

**Correspondence**
Jan Platos, Department of Computer Science, FEECS, 17. listopadu 2172/15, 708 00 Ostrava, Czech Republic.
Email: jan.platos@vsb.cz

**Funding information**
European Regional Development Fund, Grant/Award Number: Project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466); Vysoká Škola Báňská - Technická Univerzita Ostrava, Grant/Award Number: Student Grant Agency SP2019/135

## Summary

Digital steganography aims at hiding secret messages in digital data transmitted over insecure channels. The JPEG format is prevalent in digital communication, and images are often used as cover objects in digital steganography. Optimization methods can improve the properties of images with embedded secret but introduce additional computational complexity to their processing. AVX instructions available in modern CPUs are, in this work, used to accelerate data parallel operations that are part of image steganography with advanced optimizations.

**KEYWORDS**

acceleration, AVX, particle swarm optimization, permutation, steganography

## 1 | INTRODUCTION

Steganography is the art of information covering. In contrast to cryptography, which hides information by scrambling and reversible modifications of the secret but leaves the encrypted data visible, steganography seeks ways for concealing the very existence of the secret information.[1] It has a long history that can be traced to the ancient civilizations of Egypt, Greece, and China, among others.[1-3] Herodotus reported how secret messages on wooden tablets were covered by a layer of wax during wartime. In another case, the hidden information was tattooed on skin, later covered by hair, or engraved on female earrings.[2,4] In ancient China, information were sometimes hidden on thin sheets of silk and paper or baked in pastry.[3] Besides technical steganography, its linguistic variants (eg, acrostic) hide the secret into a specific pattern of letters of a cover text.[2,4]

Digital steganography refers to information hiding in digital media.[3] It aims at hiding secret messages in digital data transmitted over digital communication channels. The goal of digital steganography is to conceal both, the secret itself and the mere fact that it was sent. It facilitates covert communication by two major types of methods: data insertion, which adds additional information to the original content, and data substitution, which changes the bytes of the cover data so that it is not extended but unnoticeably modified to hold the secret.[2]

Although the primary purpose of steganography is private communication over insecure channels required by, for example, military, dissidents or criminal groups,[2] it has many other applications.[2,3,5] Digital steganography can be used for watermarking (by including hidden information about, eg, the license of data), authentication (hidden information proves the authenticity of data), tracking (hidden information identifies the true owner/author of data), and enhancement of data (including new information that could break the software directly in data),[5] fingerprinting and copy control (hiding unique identifier in each authorized copy of data),[2] and device protection (hidden information identifies authorized client of a device and/or service).[3]

Steganography and especially steganalysis, ie, the art of discovering the existence of covert information,[3] introduce additional layers of data processing and increase its overall computational costs.[2,4,6] An efficient implementation of steganographic and steganalytic algorithms is therefore

crucial for their practical applications. Parallel and distributed computation have been successfully used to accelerate steganography[7,8] and steganalysis.[6,9,10] The acceleration aims at the computationally most expensive operations, including statistical image analysis,[10] image feature extraction,[6,9,10] classifier training,[7,10] and error correction.[8] It takes advantage of the fact that steganographic primitives such as convolution[9] are, to a large extent, data parallel and suitable for an efficient execution on data-parallel devices, including accelerators and graphic processing units (GPUs). Their high-performance parallel implementations have been shown to yield a speedup of one to three orders of magnitude.[9,10]

Advanced vector extensions (AVXs) are a set of instructions for single instruction multiple data (SIMD) operations on modern CPUs.[11] It extends previous SIMD instruction sets (MMX, SSE) by introducing 256-bit–wide SIMD registers and vector units, implementation of three- and four-operand operations, relaxed memory alignment requirements, and other improvements. The use of AVX has a positive impact on the performance of implemented algorithms. Suitable data parallel operations such as vector and matrix operations can be accelerated by the factor of two and more.[12] It can be also used to speed up stochastic methods such as Monte Carlo simulations,[13] image processing tasks,[14] etc. The use of AVX can also improve the power costs and energy efficiency of various computations.[14,15] It represents a CPU-based alternative to GPU parallelization that is implemented in modern CPUs and is, in many cases, able to outperform GPU-based solutions.[13]

In this work, the acceleration of an advanced image steganographic algorithm by the use of AVX instructions is proposed, implemented, and experimentally evaluated. The steganographic algorithm, first introduced by Li and Wang,[16] is data-driven and seeks for each cover image and each block of bits to be hidden an optimum permutation to ensure that the substitution alters only the minimum number of bits in the cover. That guarantees that the visual appearance of the cover image is affected only minimally. The search for an optimum permutation is realized by a nature-inspired optimization algorithm, particle swarm optimization (PSO). PSO is a data-parallel method, working with vectors, and can be efficiently sped-up by data-parallel hardware such as SIMD instructions available in modern processors. The proposed acceleration strategy implements all feasible operations of the PSO for image steganography in AVX and demonstrates that a careful vectorization can reduce the execution time of the algorithm on the test data on average by a factor of 5.3. This is an important result indicating that properly implemented PSO-based image steganography can be on an appropriate hardware used for practical applications.

The remainder of this article is organized as follows. Section 2 outlines the basic principles of steganography, image steganography, and optimization in steganographic applications. Section 3 briefly summarizes the basic principles of particle swarm optimization, the nature-inspired optimization algorithm used in this work. Section 4 discusses the AVX instruction set and the way it is used to accelerate the execution of the image steganographic algorithm. An experimental evaluation of the proposed method is described in Section 5. Finally, major conclusions are drawn and future work outlined in Section 6.

## 2 | STEGANOGRAPHY

Steganography deals with information hiding[1,3,4] and concealed communication.[2,4] It is a set of techniques related to watermarking,[3] information hiding, and cryptography.[1,2] There are two major families of steganographic methods. Technical steganography uses various scientific and technical methods to cover the hidden information.[3,4] It includes the use of false surfaces (wooden tablets with layers of wax), invisible inks, micro dots and micro prints, etc. Linguistic steganography, on the other hand, hides information in (written) natural language.[3] Semagrams use text properties, special visual symbols, or special signs to hide secret information. Anomalies in font size, spacing, and indentation all fall into this category as well. Open codes[3] (acrostics[4]) hide information in specific patterns of letters (eg, the second letter of each word) not apparent to unauthorized readers. Another way to hide information in text is the use of intentional errors or specific stylistic features that determine the location where is the secret information embedded.[4]

*Digital steganography*, with the goal to hide information in digital data, is an important area of modern computer security.[1-4] The complementary field of *digital steganalysis* is concerned with detecting the evidence of hidden data and its recovery[1-4] or destruction.[4] Different types of data can be used as cover objects in which the secret message is embedded for transmission over public channels.[3] The cover data is not related to the embedded content and serves only as a decoy to hide the existence of the communication.[2] Although different media can be used as cover in digital steganography, digital images[17,18] are used most frequently. Besides still images, video[19] and audio files,[20] text,[21] network protocols,[22] and properties of specific digital document formats[23] have all been successfully used to hide information in digital communications. Advanced methods such as spread-spectrum steganography[24] and DNA–based steganography[25] can be used to enhance the traditional steganography techniques and improve their properties (eg, capacity to hide content). The type of cover and the character of the secret content are crucial for the selection of suitable information embedding and recovery algorithms. However, the majority of steganographic systems share the same fundamental structure, as shown in Figure 1.

### 2.1 | JPEG-based image steganography

Image steganography is the best-known and the most used type of digital steganography. The embedding of secret message into digital images can be achieved by several high-level strategies.[17] Cover selection steganography is based on a shared database of predefined images that contain the information to be transmitted encoded in a visual symbol (eg, the presence of a particular object in the scene) or hash of the image. The information embedding and recovery algorithm is associated with the knowledge of the meaning of the symbols or the hash function. A similar image steganography strategy is based on cover construction (synthesis).[17] In this case, the cover is for every message synthesized so

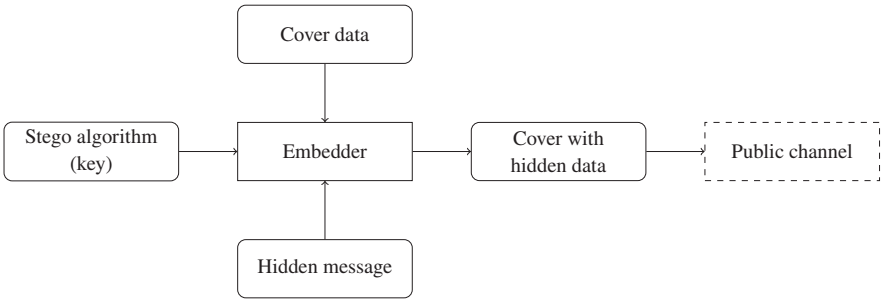



**FIGURE 1** A generic steganography system[2,17]

that it contains the required symbols and no database of existing images is required. However, the shared knowledge of the visual encoding is still expected and cover synthesis additionally relies on a convincing appearance of the constructed message (in visual or at least feature space). The most frequently used type of image steganography is based on cover modification.[17] It embeds the secret information into the cover image by subtle (visually negligible) modifications of its data. In practice, methods based on cover modification take advantage of different digital image formats (raster or vector) and exploit the properties of the image representation they use. Methods to embed hidden content into bitmap (BMP), Graphics Interchange (GIF), Portable Network Graphics (PNG), and Joint Photographic Experts Group (JPEG) formats were proposed[18] in the past and are still actively developed.[26]

The JPEG format is one of the most frequently used image formats in digital communication and on the Internet. It is based on a lossy compression algorithm that efficiently reduces the size of images with smooth changes of tone and color. Due to the ability to capture realistic images (scans, digital photography), it can be found on the majority of Web pages. The fundamental principles of the compression algorithm and the prevalence of JPEG images in digital media make them an ideal target for image steganography. The basic steps of the JPEG compression algorithm are shown in Figure 2. The compression causes information loss at several stages of the algorithm. The first loss is due to the color subsampling at the beginning of the process. Color information is in JPEG reduced by factor 4 or 2, while the brightness part is left untouched. This is done because the human eye is more sensitive to brightness than color. A minor loss of information is introduced by rounding the coefficient of the discrete cosine transform (DCT). However, the main loss of information happens during the quantization of the DCT coefficients using a quantization table. The quantization table has in the JPEG format a recommended content, but alternative quantization strategies can be used by different compression utilities. The data after the quantization is encoded using a lossless entropic coding, and therefore, any changes, ie, the addition of a hidden message, will not modify it. The standard JPEG quantization table is shown in Figure 3.

The main purpose of steganography is information hiding. Cover modification–based steganographic algorithms strive to find information embedding approaches that yield the modification of the least number of bits in the cover. The Least-Significant-Bit (LSB) embedding is the most common steganographic algorithm in this domain. It is a general and simple data embedding strategy that can be applied to any type of digital cover.[17] LSB replaces $k$ least significant bits in selected parts of the cover by bits from the secret message. The modification of the least significant bits in the cover introduce only small changes to cover data. The capacity of an image cover in a LSB-like information embedding scheme depends on its size, the number of least significant bits that are changed, and other properties of cover media format. For example, embedding into zero DCT coefficients significantly reduces the compression ratio and, potentially, introduces visible JPEG artifacts.

Most JPEG steganographic methods change only DCT coefficients that are greater than zero. According to Li and Wang,[16] the steganographic capacity of a JPEG file can be increased by modifications of the quantization table. A modified quantization table, described in the aforementioned work,[16] is shown in Figure 4. The modified table has in the upper left triangle above the secondary diagonal coefficients with a reduced value. It

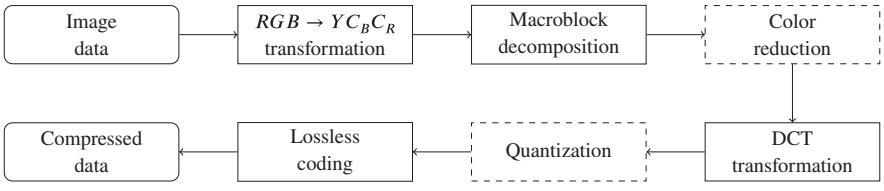



**FIGURE 2** JPEG compression algorithm; dashed rectangles highlight main data loss in the algorithm

$$\begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 59 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

**FIGURE 3** Standard quantization table of JPEG compression

$$\begin{bmatrix} 8 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 55 \\ 1 & 1 & 1 & 1 & 1 & 1 & 69 & 59 \\ 1 & 1 & 1 & 1 & 1 & 87 & 80 & 62 \\ 1 & 1 & 1 & 1 & 68 & 109 & 103 & 77 \\ 1 & 1 & 1 & 64 & 81 & 104 & 113 & 92 \\ 1 & 1 & 78 & 87 & 103 & 121 & 120 & 101 \\ 1 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

**FIGURE 4** Modified quantization table of JPEG compression

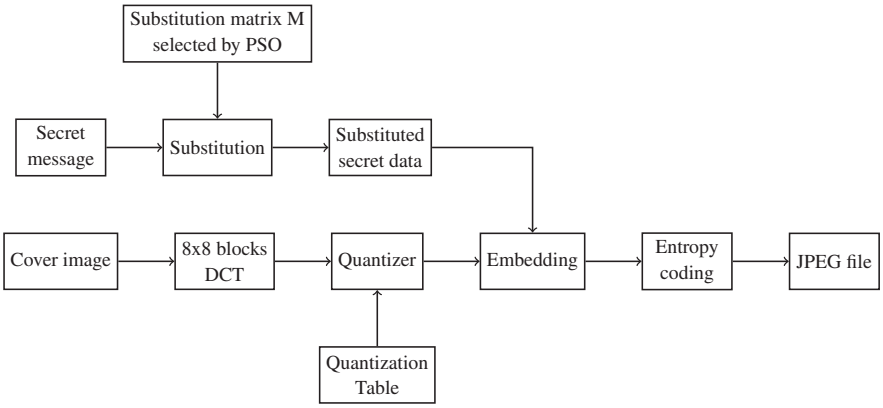



**FIGURE 5** A workflow of the embedding procedure[16]

leads to a JPEG compression with larger number of nonzero DCT coefficients and increased steganographic capacity. However, the compression ratio of the image is reduced.

Another aspect of image steganography that can be optimized is the number of bits altered by the embedding procedure in the cover image. An optimization-based method that looks for an optimum ordering of the bits in the input message was proposed in the work of Li and Wang.[16] It extends the steganographic procedure by a combinatorial optimization step that consists in the search for an optimum permutation of a set of objects. The workflow of the optimization is depicted in Figure 5.

A general combinatorial optimization problem, $\Pi = \{I, sol(i)\}_{i} i \in I\}, m\}$, can be defined as a minimization or maximization problem that consists of a set of problem instances, $I$, a set of feasible problem solutions, $sol(i)$, for every problem instance, $i \in I$, and a function, $m : \{(i,q)|i \in I, q \in sol(i)\} \rightarrow \mathbb{Q}_{+}$, where $\mathbb{Q}_{+}$ is the set of positive rational numbers and $m(i,q)$ is the value of solution, $q$, of problem instance, $i$.[27] An optimal solution to an instance of a combinatorial optimization problem is such solution that has a maximum (or minimum) value among all other solutions. Famous combinatorial optimization problems include, for example, the traveling salesman problem, the knapsack problem, and the linear ordering problem.[27]

Combinatorial optimization problems can be addressed by different types of algorithms. Bio-inspired metaheuristics are often successful in situations where exact methods are infeasible due to too high computational complexity and heuristic approaches are unknown or fail to find optimal problem solutions. On the other hand, the choice of an appropriate nature-inspired algorithm and suitable solution representation is not trivial and affects the performance and results of the optimization procedure. Particle swarm optimization[28-30] is a popular nature-inspired metaheuristic method that has shown an excellent ability to solve complex high-dimensional optimization problems. It was also successfully applied in the area of JPEG steganography.[16] Although many variants of PSO exist, the traditional PSO with inertia weight[31] is, in this work, used to solve the combinatorial optimization problem arising in JPEG steganography. The choice of the algorithm is motivated by several reasons. It is data parallel, works with candidate solutions in the form of real-valued vectors, and is, therefore, suitable for acceleration by SIMD instructions such as AVX. At the same time, it is simple and has lower overhead than more sophisticated PSO variants.[32] PSO with inertia weight is also exactly the same algorithm that was used in the steganographic system of Li and Wang.[16] Finally, the main objective of this work is the study of AVX-based vectorization strategies for image steganography utilizing the PSO rather than evaluation of different optimization methods. However, it can be noted that the proposed vectorization strategy can be easily applied steganographic systems using similar real-valued optimizers such as differential evolution,[33] self-organizing migrating algorithm,[34] artificial bee colony algorithm,[35] and the grey wolf optimizer[36] to name just a few.

## 3 | PARTICLE SWARM OPTIMIZATION

Particle swarm optimization (PSO) is a global, population-based search and optimization algorithm based on the simulation of swarming behavior of bird flocks, fish schools, and even human social groups.[28-30] PSO uses a population of motile candidate particles characterized by their position, $\mathbf{x}_i$, and velocity, $\mathbf{v}_i$, inside an $n$-dimensional search space that they collectively explore. Each particle remembers the best position (in terms of

fitness function value) it visited, $\mathbf{y}_i$, and is aware of the best position discovered so far by the entire swarm, $\hat{\mathbf{y}}$. In each iteration, the velocity, $\mathbf{v}_i$, of a particle, $i$, is updated[28] according to

$$\mathbf{v}_i^{t+1} = c_0\mathbf{v}_i^t + c_1\mathbf{r}_1^t\left(\mathbf{y}_i - \mathbf{x}_i^t\right) + c_2\mathbf{r}_2^t\left(\hat{\mathbf{y}}^t - \mathbf{x}_i^t\right), \tag{1}$$

where $c_0$ is inertia weight and $c_1$ and $c_2$ are positive acceleration constants that influence the tradeoff between exploration and exploitation. Vectors $\mathbf{r}_1$ and $\mathbf{r}_2$ contain random values sampled from a uniform distribution. The position of particle $i$ is updated given its velocity[28] as follows:

$$\mathbf{x}_i^{t+1} = \mathbf{x}_i^t + \mathbf{v}_i^{t+1}. \tag{2}$$

PSO is useful for dealing with problems whose solution can be represented as a point or surface in the $n$-dimensional search space. Candidate solutions (particles) are placed in this space and provided with a random initial velocity. The particles then move through the search space and are periodically evaluated using a fitness function. Over time, particles are accelerated toward those locations in the problem space that have relatively better fitness values. In addition to the basic model, there is a number of alternative versions of the PSO algorithm, including self-tuning PSO, niching PSO, and multiple-swarm PSO. These variants have been developed to improve the convergent properties of the algorithm or to solve other specific problems.[28,29] Successful PSO applications can be found in a many diverse domains ranging from, eg, practical control of wireless sensor networks[37] to the search of optimum parameters of mathematical models.[38]

Combinatorial optimization problems that require well-structured discrete solutions, on the other hand, represent a challenge for algorithms with real-valued candidate solutions such as PSO. The position and velocity updates of particles are in PSO performed in each dimension independently, whereas the elements of permutations are not independent on each other.[39] Nevertheless, several methods for representation of permutations for PSO have been proposed in the past.[39-41]

Hu et al[39] proposed a PSO with a modified particle velocity for permutation problems. The velocity is in the algorithm normalized to the range $[0, 1]$, and the velocity in each dimension corresponds to the probability that the position in a particle, $\mathbf{x}_i$, is exchanged with the corresponding position in the best particle, $\hat{\mathbf{y}}$. To avoid trapping in local optima, the algorithm applies a mutation operator that randomly swaps the positions in a particle. Another work proposed a different permutation encoding strategy for PSO.[40] It applied a principle called the smallest position value (SPV) rule to associate particles with permutations. The principle is identical to the random keys encoding known from the area of genetic algorithms.[42] Under the SPV rule, particle coordinates in each dimension in $\mathbf{x}_i$ are ordered from smallest to largest, and the changes in their position are associated with the changes of permutation indices. An example of the SPV rule is shown in Equation (3)

$$\pi_5 = \begin{pmatrix} 0.2 & 0.3 & 0.1 & 0.5 & 0.4 \\ 2 & 3 & 1 & 5 & 4 \end{pmatrix}. \tag{3}$$

More recently, Eddaly et al[41] developed a combinatorial PSO to solve the blocking flowshop scheduling problem. The authors proposed a system of particle updates that generates only feasible solutions (valid permutations) and showed that the combinatorial PSO is able to find good flowshop schedules.

In this work, the SPV-based permutation representation is employed to find optimum permutations of input symbols in a vectorized JPEG-based steganographic method.

## 4 | AVX INSTRUCTIONS AND THE ACCELERATION OF JPEG STEGANOGRAPHY

Intel's advanced vector extensions is a set of processor instructions introduced in the second generation of the Intel Core processor family, the Sandy Bridge microarchitecture. Since the Jaguar microarchitecture, it is also supported by AMD processors. AVX is a single instruction multiple data 256-bit instruction set for data level parallelism allowing the processing of multiple data elements simultaneously. It was designed to simplify the implementation of software with various degrees of thread parallelism and different lengths of processed data vectors. Nowadays, several generations of the AVX instruction set exist. AVX2 has sixteen 256-bit-wide registers able to operate in both 32-bit and 64-bit modes. AVX-512 expands the total number of registers to 32 and extends their width to 512 bits.

AVX instructions provide various functions: numeric, arithmetic, cryptographic, convert, etc. They can operate with single or double-precision floating-point variables and data types. The AVX supports fused multiply-add (FMA) operations on vector registers as well.[12,43] This SIMD approach can improve the performance of arbitrary applications through their data-parallel execution, but the selection of suitable algorithm, programming model, and compiler is crucial for the success of AVX-based vectorization.

In this work, an AVX vectorization of a JPEG steganography optimized by PSO is developed and evaluated. The steganographic algorithm uses PSO as an optimization procedure to search for permutations of blocks of input bits to minimize the number of modifications to the cover image. The permutation is represented by PSO particles and decoded using the smallest position value rule. The parallelization of this algorithm requires vectorization of the core PSO operations, the decoding process, and the fitness function evaluation. The PSO operations, summarized in equation (1) and equation (2), are applied to real-valued vectors (particles). They are embarrassingly parallel, and their vectorization by AVX is straightforward. The decoding process, associated with the SPV rule, requires sorting of the values in each particle, $\mathbf{x}_i$. The proposed vectorization

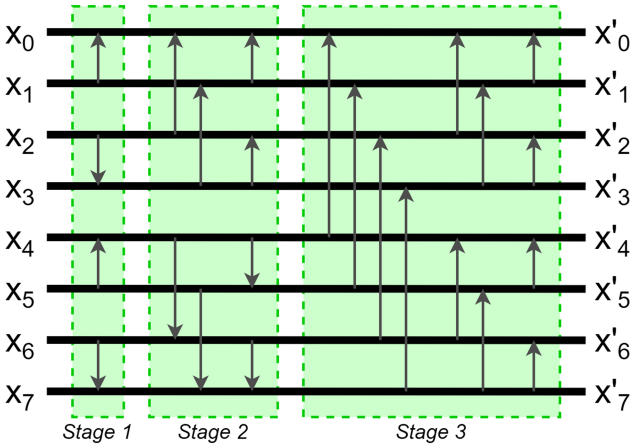



**FIGURE 6** Bitonic sorting network for 8 values

strategy implements Bitonic sort (Bitonic mergesort) as an efficient parallel sorting algorithm. The fitness function evaluation requires the computation of the Hamming distance between the permuted pairs of input bits and the DCT coefficients of the cover image.

Bitonic sort (BiS)[44] is an efficient parallel sorting algorithm. It is a sorting network with a steady delay of $O(n \log n^2)$, where $n$ is the length of the input sequence. The number of element comparisons performed by BiS is higher than that of the traditional (sequential) Mergesort, but the algorithm is more suitable for parallel execution. It compares the elements of the input data set in a predefined sequence (called sorting network) independent on their values. The sorting network, illustrated in Figure 6, is able to mutually compare all values in the input data set and sort them in the increasing (or decreasing) order. In each step, BiS performs $n/2$ element comparisons and creates $n/2$ partially sorted sequences. The sequences are, in the following steps, further sorted until a full ordering of the data set is achieved.

In the performed experiments, the PSO is used to find a permutation of 32 pairs of bits that are supposed to be hidden in the cover image (ie, the dimension of the solution space is 32). The core PSO operations, velocity and position updates, are data parallel, and their AVX implementation is straightforward. In order to maximize the efficiency of the vectorization, the crucial parts of the code have been implemented using Intel's AVX Intrinsics. The use of AVX2 instructions can, for PSO in a 32-dimensional search space, provide a theoretical speedup by the factor of 8. However, the overhead (eg, register loads and stores) and data dependencies reduce the real effect of the AVX-based vectorization to a lower factor (on average, 5.3 in this work).

The sequential, nonoptimized, version of the steganographic algorithm uses a sequential implementation of the bubble sort algorithm to decode the permutations and the XOR operation together with bit shifts to evaluate the fitness function, ie, the Hamming distance between DCT coefficients of the cover image and the permuted input data.

The vectorized version of the algorithm works with single precision floating point data types whenever possible. Following this approach, the complete permutation of 32 values can fit up to only four AVX registers without any loss of accuracy. Then, an AVX implementation of Bitonic sort is used to decode the permutation, ie, to find the indices of the corresponding permutation. There is also an inline assembly code responsible for a fast conversion of the AVX register to a vector of integers. The performance effects of each of the optimization step, described above, are analyzed in the experimental part of this work. The final, most optimized, parallel version of the algorithm also benefits from a number of other small optimizations, improving its performance that are beyond the scope of this article. Most of them improve the efficiency of memory accesses and data manipulations.

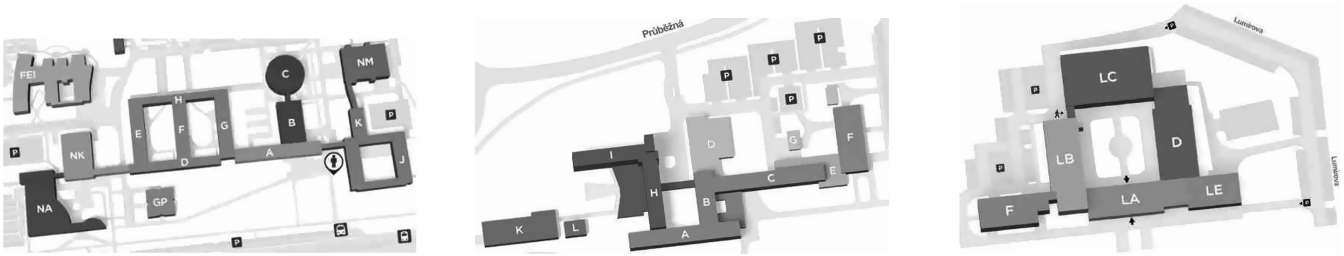
## 5 | EXPERIMENTS

A series of computational experiments was carried out in order to evaluate the proposed vectorization strategies. The experiments consisted in the search for an optimum permutation of pairs of input bits (from the hidden message) by PSO.

Three hidden messages were used in the experiment. They contain grayscale images of parts of street plan of the campus of VSB-Technical University of Ostrava. The hidden images are shown in Figure 7 The famous Lena image was used as a cover image in which the hidden messages were embedded.

The first series of experiments focused on the ability of PSO to find an optimum permutation of pairs of input bits from the hidden message. The cover image was divided into 1024 8x8 blocks, and each block was represented by 36 discrete cosine transform (DCT) coefficients. The input message was embedded into the first 32 DCT coefficients. The pairs of input bits were encoded into two least significant bits of the DCT coefficients in each block, and the PSO looked for a permutation of the sequence of input bits, $\pi_{32}$, so that the Hamming distance between the sequence of LSBs and the permuted input bits was minimized. The permutation was sought for each block of input data independently, and the experiments were, due to the stochastic nature of PSO, repeated 50 times independently.

The algorithm, used in this experiment, was a global best PSO with SPV-based permutation representation and Hamming distance between permuted input sequence and LSB sequence as fitness function. The average Hamming distance between DCT coefficients and input bit sequence




(A) (B) (C)

**FIGURE 7** Hidden images. A, Hidden image H1 (9137 bytes); B, Hidden image H2 (9139 bytes); C, Hidden image H3 (8679 bytes)

**TABLE 1** JPEG-based steganography optimized by PSO

| Hidden image | Average Hamming distance | | Average |
|---|---|---|---|
| | Without optimization | With optimization | improvement [%] |
| H1 | 24.253 | 11.104 | 54.217 |
| H2 | 24.624 | 12.335 | 49.906 |
| H3 | 24.647 | 12.485 | 49.344 |

**TABLE 2** Results of the PSO evolution

| Hidden image | Improvement [%] | | |
|---|---|---|---|
| | Min. | Mean ($\sigma$) | Max. |
| H1 | 53.715 | 54.217 (0.237) | 54.821 |
| H2 | 49.503 | 49.906 (0.175) | 50.304 |
| H3 | 49.037 | 49.344 (0.130) | 49.643 |

with and without optimization of PSO is summarized in Table 1. The table clearly illustrates that PSO is able to find permutations that reduce the Hamming distance between DCT coefficients and bits in the input sequence by around 50 percent on average. The analysis of the 50 independent PSO runs for all test images is shown in Table 2. It shows that the PSO was, for each test image, able to obtain an improvement of at least 49%. The best reduction in the Hamming distance, 54.8%, was achieved for the test image H1. The standard deviation of the Hamming distance reduction was for each test hidden image lower than 0.25%. That indicates that the proposed optimization approach is robust, and independent PSO runs are able to find permutations with similar quality.

## 5.1 | AVX optimization of the algorithm

In the second set of experiments, five implementations of the algorithm with different level of optimization were compared to evaluate the potential of vectorization to accelerate JPEG steganography with PSO optimization. The computational experiments were executed on a server with Intel(R) Core(TM) i7-4770K Haswell CPU running at 3.5 GHz with 4 x 32 KB L1 cache, 4 x 256 KB L2 cache, and 8 MB L3 cache. All experiments were limited to single-core execution, using GNU compiler collection g++ version 6.3.0 with aggressive optimization *-O3*. All compiler parameters are shown in Listing 1. The GCC compiler achieved the best performance among the C++ compilers available at the time of the evaluation. For a further investigation of the performance of the algorithm, we use the Intel(R) Parallel Studio tools. The whole code was limited to execute strictly on a single core to measure and tune the maximal potential of the execution. The further speedup is possible by multithreaded execution, depending on the number of cores used.

The optimization of the PSO algorithm consists of several parts. First, there is the PSO part responsible for finding the best possible permutation. We had the experience of boosting the performance of PSO from the previous research. The research confirmed that there is a significant speedup in algorithm performance compared to its scalar version.

For the optimization of steganography code, several versions of the code with different level of optimization were prepared. All versions (labeled A - E) have the same PSO parameters (see Table 3) and share these properties: they were all optimized by the compiler for speed (O3) and use auto-vectorization wherever possible. The code is a single-threaded implementation of the algorithm written in C++.

The PSO was executed with a variable number of particles (100, 200, 400, 800, and 1600) and terminated after 1000 iterations. All remaining PSO parameters remained fixed during all experiments to provide consistent results. For each configuration, 50 experimental runs were observed. Each experiment was repeated in similar conditions to acquire independent results. During experiments, we try to mitigate all potential distortion from the OS and other software. The results are stable with a low standard deviation (ie, 8.56e-04 for experiments with version A).

```
-std=c++1y -O3 -mtune=core-avx2 -march=core-avx2 -ffast-math -ffinite-math-only -ftree-vectorize
    -fno-unroll-loops -g -c -fmessage-length=0 -mavx2
```

**Listing 1** G++ compiler options

**TABLE 3** PSO parameters involved in all experiments

| Param | Value |
|---|---|
| PSO iterations | 1000 |
| Particles in swarm | 100, 200, 400, 800, 1600 |
| Number of dimensions | 32 |
| Inertia weight (c0) | 0.729 |
| Local weight (c1) | 1.49445 |
| Global weight (c2) | 1.49445 |

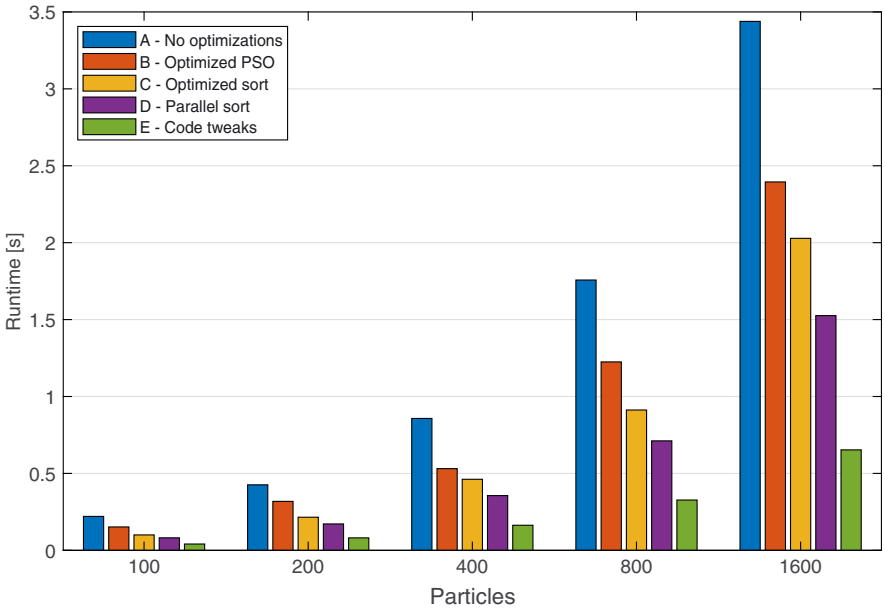



**Figure 8** Execution time of the investigated code versions with various level of optimization

**TABLE 4** Execution time (in seconds) of all code versions for PSO with 400 particles

| | Min | Mean ($\sigma$) | Max |
|---|---|---|---|
| **A** | 0.8560 | 0.8574 (0.0013) | 0.8610 |
| **B** | 0.5290 | 0.5311 (0.0029) | 0.5390 |
| **C** | 0.4610 | 0.4619 (0.0009) | 0.4640 |
| **D** | 0.3550 | 0.3558 (0.0009) | 0.3580 |
| **E** | 0.1630 | 0.1630 (0.0000) | 0.1630 |

The performance of all code versions is illustrated in Figure 8. A detailed comparison of all versions for PSO with 400 particles is then summarized in Table 4. An ANOVA test confirms that there is a statistically significant difference in the performance of the code with different levels of optimization at significance level $\alpha = 0.05$ (details are shown in Table 5).

The first version, A, has no manual optimizations whatsoever. It is an auto-vectorized version of the scalar PSO. The fitness function relies on naïve implementation of the bubble sort algorithm (Listing 2) for a simple decoding of the PSO particle to the permutation form. The fitness is

**TABLE 5** Runtime of ANOVA for PSO version with various number of PSO particles

| Particles | F | F crit | P-value |
|---|---|---|---|
| 100 | 97858,7357 | 2,5787 | 5,3993E-88 |
| 200 | 154705,1932 | 2,5787 | 1,8088E-92 |
| 400 | 54993,7941 | 2,5787 | 2,3059E-82 |
| 800 | 183251,9332 | 2,5787 | 4,0067E-94 |
| 1600 | 19307,4788 | 2,5787 | 3,8658E-72 |

```
do {
  changed = false;
  for (unsigned int i = 0; i < n - 1; i++){
    if (key[index[i]] > key[index[i + 1]]){
      int tmp = index[i];
      index[i] = index[i + 1];
      index[i + 1] = tmp;
      changed = true;
    }
  }
}while (changed);
```

**Listing 2** Bubble sort

```
for (unsigned int i = 0; i < messageLength); i++) {
  diff = lena_data[0][i] ^ message[index[i]];
  error += diff & 1;
  error += (diff >> 1) & 1;
}
```

**Listing 3** Bit counter - naive

evaluated by a bit counter comparing the last two bits (Listing 3) of source picture and hidden message. This version sets the baseline performance to which all manually vectorized versions are compared.

The second version, named B, uses AVX optimized PSO algorithm without optimizations in the fitness function. The rest of the code is auto-vectorized like in the previous version, A. The third version, C, uses an optimized PSO and standard library sort instead of the naïve bubble sort implementation. The efficiency of the bit counter was optimized as well using *popcnt* function, see Listing 4. Version C outperforms A and B in all experimental configurations. However, the sorting procedure can be further enhanced by the use of a special parallel sorting algorithm (Listing 5). Bramas [45] showed that parallel bitonic sort can be implemented very efficiently in AVX.

Version D adopts bitonic sort as the algorithm to decode PSO particles to permutations. This version also uses all other previous manual optimizations. In fact, it corresponds to A but applies SIMD optimizations to all parts of the code that manipulate permutation data. Nevertheless, the results show more than two times faster execution time than that of A. If we compare it with naïve serial bubble sort function in version A, the overall speedup is of factor 11.5 (just for the function itself).

In the last version, E, the whole codebase of D was revised, and all remaining potential bottlenecks were reimplemented for maximum performance. The bit counter was rewritten to use precomputed values (see Listing 6), permutation indexing rely on AVX instructions and assembly code. Several other minor small performance tweaks were used. The resulting code is more than five times faster than the initial auto-vectorized version A.

Table 6 compares the performance of the least and most optimized version (A and E) with respect to the number of particles. The final optimized version is in average 5.3 times faster (81.15% improvement) than the nonoptimized version.

```
for (unsigned int i = 0; i < messageLength; i++) {
  error += _mm_popcnt_u32( (lena_data[0][i] ^ message[
      index[i]]) & 0x3u);
}
```

**Listing 4** Bit counter - popcnt

```
inline void CoreSmallSort2(__m256& input, __m256& input2){
{
  __m256i idxNoNeigh = _mm256_set_epi32(6, 7, 4, 5, 2, 3, 0, 1);
  __m256 permNeigh = _mm256_permutevar8x32_ps(input, idxNoNeigh);
  __m256 permNeigh2 = _mm256_permutevar8x32_ps(input2, idxNoNeigh);
  __m256 permNeighMin = _mm256_min_ps(permNeigh, input);
  __m256 permNeighMin2 = _mm256_min_ps(permNeigh2, input2);
  __m256 permNeighMax = _mm256_max_ps(permNeigh, input);
  __m256 permNeighMax2 = _mm256_max_ps(permNeigh2, input2);
  input = _mm256_blend_ps(permNeighMin, permNeighMax, 0xAA);
  input2 = _mm256_blend_ps(permNeighMin2, permNeighMax2, 0xAA);
}
--- other parts omitted ---
```

**Listing 5** Bitonic sort - part of sorting network

```
const char precError [4] = {0,1,1,2};
for (unsigned int i = 0; i < messageLength; i++) {
  error += precError[(lena_data[0][i] ^ message[index[
      i]]) & 0x3u];
}
```

**Listing 6** Bit counter - precomputed array

**TABLE 6** Execution time (in seconds) of the least (A) and the most (E) optimized code version

| Particles | A | | | E | | |
|---|---|---|---|---|---|---|
| | Min | Mean ($\sigma$) | Max | Min | Mean ($\sigma$) | Max |
| **100** | 0.2200 | 0.2205 (0.0005) | 0.2210 | 0.0410 | 0.0410 (0.0000) | 0.0410 |
| **200** | 0.4250 | 0.4258 (0.0004) | 0.4260 | 0.0810 | 0.0816 (0.0009) | 0.0840 |
| **400** | 0.8560 | 0.8574 (0.0013) | 0.8610 | 0.1630 | 0.1630 (0.0000) | 0.1630 |
| **800** | 1.7560 | 1.7572 (0.0006) | 1.7580 | 0.3270 | 0.3271 (0.0003) | 0.3280 |
| **1600** | 3.4360 | 3.4386 (0.0013) | 3.4410 | 0.6530 | 0.6531 (0.0003) | 0.6540 |

## 6 | CONCLUSIONS

Image steganography is a popular data hiding strategy that can be used to conceal private communication in unsecure digital channels. Optimization-based methods improve various properties of steganographic algorithms. They can be employed to, eg, increase the capacity of the cover or to minimize the number of modifications of the cover object. The latter is especially crucial for digital image steganography, which can be compromised by an automated or manual visual analysis of the cover. Efficient data embedding and recovery are essential for practical applications of digital steganography, especially when optimization takes place during the data embedding procedure. AVX instructions form a family of instruction sets for efficient data parallel (SIMD) operations introduced by Intel and available in many modern CPUs. In this work, AVX was used to achieve a vectorized implementation of a PSO-based JPEG steganographic algorithm.

Instead of PSO, we could use any optimization algorithm to achieve similar results for finding the best possible combination of steganography permutations. However, with the parallel computation in mind, we choose to use the PSO because of its high potential for parallelization.

Computational experiments show that different levels of code optimizations (auto-vectorization vs hand-tuned optimizations) improve the performance of the code, and careful manual optimization can accelerate the execution of the algorithm in average 5.3 times. Even if the full performance impact on single execution time is low and saves just a few seconds, we need to remember that the steganographic algorithm executes the PSO several times and thus gains significant performance boost.

### ACKNOWLEDGMENTS

This work was supported by the European Regional Development Fund under the project AI & Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15 003/0000466) and by the project SP2019/135 of the Student Grant System, VSB – Technical University of Ostrava.

### ORCID

*Vaclav Snasel* https://orcid.org/0000-0002-9600-8319
*Pavel Kromer* https://orcid.org/0000-0001-8428-3332
*Jakub Safarik* https://orcid.org/0000-0002-3360-2302
*Jan Platos* https://orcid.org/0000-0002-8481-0136

### REFERENCES

1. Raggo M, Hosmer C. *Data Hiding: Exposing Concealed Data in Multimedia, Operating Systems, Mobile Devices and Network Protocols*. Waltham, MA: Elsevier Science; 2012.
2. Cox I, Miller M, Bloom J, Fridrich J, Kalker T. *Digital Watermarking and Steganography*. Burlington, MA: Elsevier Science; 2007.
3. Shih F. *Digital watermarking and Steganography: Fundamentals and Techniques*. Boca Raton, FL: CRC Press; 2017.
4. Katzenbeisser S, Petitcolas F. *Information Hiding*. Norwood, MA: Artech House Publishers; 2016.
5. Wayner P. *Disappearing Cryptography: Information hiding: Steganography & watermarking*. 3rd ed. San Francisco, CA : Morgan Kaufmann Publishers Inc; 2009.
6. Xia C, Guan Q, Zhao X, Zhao C. Highly accurate real-time image steganalysis based on GPU. *J Real-Time Image Proc*. 2018;14(1):223-236.
7. Hu D, Wang L, Jiang W, Zheng S, Li B. A novel image steganography method via deep convolutional generative adversarial networks. *IEEE Access*. 2018;6:38303-38314.
8. Bhattacharjee S, Chakkaravarthy M, Midhun Chakkaravarthy D. GPU-based integrated security system for minimizing data loss in big data transmission. In: Balas VE, Sharma Na, Chakrabarti A, eds. *Data Management, Analytics and Innovation*. Singapore: Springer Singapore; 2019:421-435.
9. Ker AD. Implementing the projected spatial rich features on a GPU. Paper presented at: Media Watermarking, Security, and Forensics; 2014; San Francisco, CA.
10. Tiwary M, Priyadarshini R, Misra R. A faster and intelligent steganography detection using graphics processing unit in cloud. Paper presented at: International Conference on High Performance Computing and Applications; 2014; Bologna, Italy.
11. Lomont C. Introduction to Intel Advanced Vector Extensions. Intel White Paper. 2011.
12. Hassan SA, Hemeida A, Mahmoud MMM. Performance evaluation of matrix-matrix multiplications using Intel's advanced vector extensions (AVX). *Microprocess Microsyst* 2016;47(PB):369-374. https://doi.org/10.1016/j.micpro.2016.10.002
13. Liyanage D, Fernando P, Mampitiya Arachchi D, Karunathilaka R, Perera A. Utilizing Intel advanced vector extensions for Monte Carlo simulation based value at risk computation. *Procedia Comput Sci*. 2017;108:626-634.
14. Mitra G, Johnston B, Rendell AP, McCreath E, Zhou J. Use of SIMD vector operations to accelerate application code performance on low-powered arm and Intel platforms. Paper presented at: IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum; 2013; Cambridge, MA.
15. Cebrián JM, Natvig L, Meyer JC. Improving energy efficiency through parallelization and vectorization on Intel core i5 and i7 processors. Paper presented at: International Conference for High Performance Computing, Networking, Storage and Analysis; 2012; Salt Lake City, UT.
16. Li X, Wang J. A steganographic method based upon JPEG and particle swarm optimization algorithm. *Inf Sci*. 2007;177(15):3099-3109. https://doi.org/10.1016/j.ins.2007.02.008
17. Fridrich J. *Steganography in Digital Media: Principles, Algorithms, and Applications*. 1st ed. New York, NY: Cambridge University Press; 2009.

18. Cheddad A, Condell J, Curran K, Kevitt PM. Digital image steganography: survey and analysis of current methods. *Signal Process*. 2010;90(3):727-752. https://doi.org/10.1016/j.sigpro.2009.08.010
19. Sadek MM, Khalifa AS, Mostafa MGM. Video steganography: a comprehensive review. *Multimed Tools Appl*. 2015;74(17):7063-7094. https://doi.org/10.1007/s11042-014-1952-z
20. Gopalan K. Audio steganography using bit modification. Paper presented at: IEEE International Conference on Acoustics, Speech, and Signal Processing; 2003; Hong Kong, China.
21. Chapman M, Davida G, Rennhard M. A practical and effective approach to large-scale automated linguistic steganography. Paper presented at: International Conference on Information Security; 2001; Málaga, Spain.
22. Zander S, Armitage G, Branch P. A survey of covert channels and countermeasures in computer network protocols. *IEEE Commun Surv Tutor*. 2007;9(3):44-57. https://doi.org/10.1109/COMST.2007.4317620
23. Bhaya W, Rahma A, Al-Nasrawi D. Text steganography based on font type in MS-word documents. *J Comput Sci*. 2013;9(7):898-904. https://doi.org/10.3844/jcssp.2013.898.904.
24. Marvel LM, Boncelet CG, Retter CT. Spread spectrum image steganography. *IEEE Trans Image Process*. 1999;8(8):1075-1083.
25. Leier A, Richter C, Banzhaf W, Rauhe H. Cryptography with DNA binary strands. *Biosyst*. 2000;57(1):13-22. https://doi.org/10.1016/S0303-2647(00)00083-6
26. Bao Z, Luo X, Zhang Y, Yang C, Liu F. A robust image steganography on resisting JPEG compression with no side information. *IETE Tech Rev*. 2018;35:1-10.
27. Jongen HT, Meer K, Triesch E. *Optimization Theory*. Berlin, Germany: Springer Science & Business Media; 2004.
28. Engelbrecht A. *Computational Intelligence: An Introduction*. 2nd ed. New York, NY: Wiley; 2007.
29. Clerc M. *Particle Swarm Optimization*. London, UK: ISTE; 2010.
30. Kennedy J, Eberhart RC. Particle swarm optimization. Paper presented at: IEEE International Conference on Neural Networks; 1995; Perth, Australia.
31. Shi Y, Eberhart R. A modified particle swarm optimizer. Paper presented at: IEEE World Congress on Computational Intelligence; 1998; Anchorage, AK.
32. van den Bergh F, Engelbrecht AP. A convergence proof for the particle swarm optimiser. *Fundam Inf*. 2010;105(4):341-374.
33. Price KV, Storn RM, Lampinen JA. *Differential Evolution: A Practical Approach to Global Optimization*. Berlin, Germany: Springer-Verlag; 2005.
34. Zelinka I. Soma — self-organizing migrating algorithm. In: *New Optimization Techniques in Engineering*. Berlin, Heidelberg: Springer Berlin Heidelberg; 2004.
35. Karaboga D, Akay B. A comparative study of artificial bee colony algorithm. *Appl Math Comput*. 2009;214(1):108-132. https://doi.org/10.1016/j.amc.2009.03.090
36. Mirjalili S, Mirjalili SM, Lewis A. Grey wolf optimizer. *Adv Eng Softw*. 2014;69:46-61. https://doi.org/10.1016/j.advengsoft.2013.12.007
37. Krömer P, Musilek P. Bio-inspired routing strategies for wireless sensor networks. In: Krömer P, Fay D, Gabryś B, eds. *Propagation Phenomena in Real World Networks*. Cham: Springer International Publishing; 2015:155-181.
38. Heckenbergerova J, Musilek P, Krömer P. Optimization of wind direction distribution parameters using particle swarm optimization. Paper presented at: Afro-European Conference for Industrial Advancement; 2015; Addis Ababa, Ethiopia.
39. Hu X, Eberhart RC, Shi Y. Swarm intelligence for permutation optimization: a case study of n-queens problem. Paper presented at: IEEE Swarm Intelligence Symposium; 2003; Indianapolis, IN.
40. Tasgetiren MF, Liang Y-C, Sevkli M, Gencyilmaz G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. *Eur J Oper Res*. 2007;177(3):1930-1947.
41. Eddaly M, Jarboui B, Siarry P. Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem. *J Comput Design Eng*. 2016;3(4):295-311.
42. Snyder LV, Daskin MS. A random-key genetic algorithm for the generalized traveling salesman problem. *Eur J Oper Res*. 2006;174(1):38-53.
43. Gepner P. Using AVX2 instruction set to increase performance of high performance computing code. *Comput Infor*. 2017;36(5):1001-1018.
44. Batcher KE. Sorting networks and their applications. Paper presented at: AFIPS '68 Fall Joint Computer Conference; 1968; Washington, DC.
45. Bramas B. A novel hybrid quicksort algorithm vectorized using AVX-512 on Intel Skylake. *Int J Adv Comput Sci Appl*. 2017;8(10):337-344. https://doi.org/10.14569/IJACSA.2017.081044


**How to cite this article:** Snasel V, Kromer P, Safarik J, Platos J. JPEG steganography with particle swarm optimization accelerated by AVX. *Concurrency Computat Pract Exper*. 2020;32:e5448. https://doi.org/10.1002/cpe.5448