Applied Soft Computing 59 (2017) 475–486

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Designing bijective S-boxes using Algorithm Portfolios with limited time budgets

Dimitris Souravlias [a], Konstantinos E. Parsopoulos [a,*], Gerasimos C. Meletiou [b]

[a] Department of Computer Science and Engineering, University of Ioannina, GR-45110 Ioannina, Greece
[b] Department of Agricultural Technology, Technological Education Institute of Epirus, GR-47100 Arta, Greece

ARTICLE INFO

Article history:
Received 5 October 2016
Received in revised form 13 April 2017
Accepted 26 May 2017
Available online 3 June 2017

Keywords:
Algorithm Portfolios
S-boxes
Optimization
Heuristics
Cryptography

ABSTRACT

Substitution boxes (S-boxes) are essential parts of symmetric-key cryptosystems. Designing S-boxes with satisfactory nonlinearity and autocorrelation properties is a challenging task for both theoretical algebraic methods and computational optimization algorithms. Algorithm Portfolios (APs) are algorithmic schemes where multiple copies of the same algorithm or different algorithms share the available computational resources, running concurrently or interchangeably on a number of processors. Recently, APs have gained increasing attention due to their remarkable efficiency in multidisciplinary applications. The present work is a preliminary study of parallel APs on the bijective S-boxes design problem. The proposed APs comprise two state-of-the-art heuristic algorithms, namely Simulated Annealing and Tabu Search, and they are parallelized according to the master-slave model without exchange of information among the constituent algorithms. The proposed APs are experimentally assessed on typical problem instances under limited time budgets. Different aspects of their performance is analyzed, suggesting that the considered APs are competitive in terms of solution quality and running time against their constituent algorithms as well as different approaches.

1. Introduction

Substitution boxes (S-boxes) constitute essential parts of modern cryptographic applications. In essence, S-boxes are multi-input multi-ouput Boolean functions that map binary input to binary output values. S-boxes lie at the core of symmetric-key cryptographic algorithms. Specifically, they are used to conceal the relation between the input key and the encrypted output message. Therefore, they have crucial impact on the algorithm's security quality [1,2].

The design of suitable S-boxes has been an active research area for several decades with significant applications in symmetric-key cryptography standards. A widely known example is the Data Encryption Standard (DES), which was introduced in 1977 and it is based on eight 6-input 4-output S-boxes [3]. DES has been proved to be vulnerable to crack attacks such as linear cryptanalysis and parallel brute-forcing. Thus, it was superseded by the Advanced Encryption Standard (AES), which was proposed in 2000 and its implementation is also based on S-boxes [4]. Specifically, it employs a properly designed Rijndael S-box that is resistant to linear [5] and differential [6] cryptanalysis, which are most popular cipher-attacks (different approaches can be found in [7]). Today, AES is widely considered as a highly secure standard.

The problem of designing S-boxes with desirable properties has been tackled both through algebraic techniques [8,9] and computational methods [10–12]. Among them, a large body of work has been devoted to the application of metaheuristic algorithms. Typical examples are Evolutionary Algorithms and Swarm Intelligence approaches, such as Particle Swarm Optimization [10], Differential Evolution [10], and Genetic Algorithms [11]. Also, trajectory-based algorithms such as Simulated Annealing have been used [12]. Despite the reported effectiveness of these algorithms, their running time efficiency has been habitually neglected in the relevant studies.

This fact can raise concerns, since most of the studied computational algorithms require a remarkably large amount of time to produce S-boxes of high nonlinearity and low autocorrelation. Indicatively, running time has been reported to reach even several days of uninterrupted execution of the algorithm for some approaches [12]. In addition, formal experimental and statistical analysis of the algorithms' performance has been rarely reported in relevant literature. Instead, most efforts have been concentrated on reporting an S-box of good quality. Hence, existing performance

* Corresponding author.
E-mail addresses: dsouravl@cse.uoi.gr (D. Souravlias), kostasp@cse.uoi.gr (K.E. Parsopoulos), gmelet@teiep.gr (G.C. Meletiou).

http://dx.doi.org/10.1016/j.asoc.2017.05.052
1568-4946/

assessments and comparisons of different metaheuristics on the S-boxes design problem are rather obscure. Nevertheless, it is widely perceived that the necessity for efficient and effective algorithms is indispensable for the specific problem type.

Algorithm Portfolios (APs) constitute algorithmic frameworks that employ a number of copies of the same algorithm (*homogeneous APs*) or different algorithms (*heterogeneous APs*) running on the same or different processors [13,14]. APs emerged several years ago as a promising algorithmic scheme to tackle challenging optimization problems. Up-to-date, there is a multitude of works that investigate their performance from different perspectives [15–22].

In early AP models, the constituent algorithms were alternately executed on a single processor by allowing to switch execution from one algorithm to another within specific time intervals [13,14]. More sophisticated resource allocation schemes have been recently proposed to determine the execution order of the algorithms on a single processor for single-objective [17] and multi-objective [20] optimization problems. Such schemes operate during the optimization process and employ prediction metrics to decide which algorithm is the most beneficial to run for the subsequent time interval. Additionally, research effort has been paid on the selection of suitable constituent algorithms for the AP [18]. Usually, this is an offline procedure that is based on the evaluation of a number of candidate algorithms under different performance metrics.

The present work constitutes a first study of parallel APs in the design of S-boxes. Previous research suggested that *hill-climbing* properties can be highly beneficial for metaheuristics in the detection of S-boxes with desirable properties [23]. Thus, a simple and efficient parallel AP approach that comprises the well studied Tabu Search (TS) [24] and Simulated Annealing (SA) [25] algorithms is considered in our study. The selection of the specific constituent algorithms is motivated by their recognized efficiency and their inherent hill-climbing capabilities. To the best of the authors' knowledge, this is the first time that TS as well as the APs approach are used for the specific problem.

The proposed algorithmic approaches are evaluated and statistically analyzed on widely used problem instances. In order to make the problem even more challenging, tight running time constraints are considered. Thus, in a first experimentation phase we apply and analyze the plain sequential TS and SA algorithms on the S-boxes design problem. The use of sequential algorithms is the most frequent approach reported in relevant literature. In a second experimentation phase, homogeneous and heterogeneous APs are composed using combinations of different TS and SA variants. Then, the capability of the APs in outperforming TS and SA with respect to solution quality and time efficiency is experimentally studied.

The considered AP model is inherently parallel, i.e., the constituent algorithms are simultaneously executed on multiple processors. In order to perform fair comparisons with the sequential algorithms, which are executed on a single processor, the parallel AP is assigned exactly the same total running time with the sequential approaches. This time is then equally allocated to the AP's constituent algorithms. Thus, the total running time of the AP cannot exceed that of the sequential algorithms; it is simply exploited differently. The obtained experimental results suggest that the simultaneous execution of multiple algorithms can improve performance in terms of solution quality and time efficiency. Also, various speculations regarding the dynamics of the APs and the TS and SA algorithms are verified on the investigated problems.

The rest of the paper is structured as follows: Section 2 offers the mathematical formulation of the problem, and Section 3 describes the employed algorithms. Section 4 is devoted to experimental configuration and analysis of the results. Finally, Section 5 concludes the paper.

2. Problem formulation

In this section, we provide the mathematical formulation of the S-boxes design problem. Let

$$f : \mathbf{B}^n \rightarrow \mathbf{B}^m,\ \mathbf{B} = \{0, 1\},$$

be a *vectorial Boolean function* that maps n Boolean input values to m Boolean output values. This function is called an n-input m-output S-box or, simply, an $n \times m$ S-box. In case of a single output, i.e., $m = 1$, the S-box degenerates to a Boolean function. The following definitions refer to Boolean function properties.

Definition 2.1. *Polarity truth table*: Let $x \in \mathbf{B}^n$. A useful representation of a Boolean function is the polarity truth table defined as

$$\hat{f}(x) = (-1)^{f(x)},$$

which maps the output values of the Boolean function from the set $\{0, 1\}$ to the set $\{-1, 1\}$.

Definition 2.2. *Linear Boolean function*: Let $x, w \in \mathbf{B}^n$. A linear Boolean function is defined as

$$L_w(x) = w_1x_1 \oplus w_2x_2 \oplus \cdots \oplus w_nx_n,$$

where w_ix_i denotes the bitwise AND operation for all $i \in \{1, 2, \ldots, n\}$, and $\oplus$ denotes the bitwise XOR. A linear Boolean function in polarity form is denoted as $\hat{L}_w(x)$.

Definition 2.3. *Affine Boolean function*: The set of affine Boolean functions includes the set of linear Boolean functions and their complements, i.e., all functions of the form

$$A_{w,c}(x) = L_w(x) \oplus c,\ c \in \mathbf{B}.$$

Definition 2.4. *Walsh–Hadamard transform of Boolean function*: The Walsh–Hadamard transform (WHT) of a Boolean function f is defined as

$$\hat{F}_f(w) = \sum_{x \in \mathbf{B}^n} \hat{f}(x)\hat{L}_w(x),$$

and it measures the correlation between the Boolean function f and the linear Boolean function $\hat{L}_w$ with $x \in \mathbf{B}^n$.

Definition 2.5. *Nonlinearity of Boolean function*: The nonlinearity NL_f of a Boolean function f is defined as

$$NL_f = \frac{1}{2}\left(2^n - WH_{\max}(f, w)\right),$$

where

$$WH_{\max}(f, w) = \max_{w \in \mathbf{B}^n} \left|\hat{F}_f(w)\right|,$$

i.e., it is the maximum absolute value of WHT. Note that the computation of $WH_{\max}(f, w)$, except for the Boolean function f, also requires to specify a linear function through the vector w.

Definition 2.6. *Autocorrelation of Boolean function*: The autocorrelation of a Boolean function f measures its self-similarity. It is defined as

$$\hat{r}_f(s) = \sum_{x \in \mathbf{B}^n} \hat{f}(x)\hat{f}(x \oplus s),$$

where $s \in \mathbf{B}^n$. The maximum absolute value of the autocorrelation of a function f is defined as

$$A\ \ C_f = \max_{s \in \mathbf{B}^n \setminus \{0^n\}} \left| \sum_{x \in \mathbf{B}^n} \hat{f}(x)\hat{f}(x \oplus s) \right|,$$

where 0^n denotes the null vector over $\mathbf{B}^n$.

Theorem 2.1. *Parseval's theorem: It holds that*

$$\sum_{w \in \mathbf{B}^n} \left(\hat{F}_f(w)\right)^2 = 2^{2n},$$

which in turn results in $WH_{\max}(f, w) \geqslant 2^{n/2}$.

Definition 2.7. *Balanced Boolean function*: If the number of 0-valued outputs of a Boolean function is equal to the number of its 1-valued outputs, then the Boolean function is called balanced.

Boolean functions compose S-boxes. An S-box, $f: \mathbf{B}^n \to \mathbf{B}^m$, is a combination of m single-output Boolean functions. In order to extend the theoretical background from Boolean functions to S-boxes, the m output values of the S-box can be transformed into a single-output Boolean function. Let $f_\beta(x)$, $\beta \in \mathbf{B}^m$, be a linear combination of the m output values of the S-box f

$$f_\beta(x) = \beta_1 f_1(x) \oplus \beta_2 f_2(x) \oplus \cdots \oplus \beta_m f_m(x),$$

where $f_i(x)$, $i \in \{1, 2, \ldots, m\}$, denotes the ith output bit of the S-box. There are $2^m - 1$ such linear combinations, excluding the trivial case of the linear combination with the null vector. We can extend the aforementioned nonlinearity and autocorrelation properties of Boolean functions to S-boxes as follows.

Definition 2.8. *Walsh–Hadamard transform of S-box*: The Walsh–Hadamard transform (WHT) of an S-box is defined as

$$\hat{F}_\beta(w) = \sum_{x \in \mathbf{B}^n} \hat{f}_\beta(x)\hat{L}_w(x). \tag{1}$$

Definition 2.9. *Nonlinearity of S-box*: The nonlinearity NL_f of an S-box is defined as

$$NL_f = \frac{1}{2}\left(2^n - WH_{\max}\left(f_\beta, w\right)\right), \tag{2}$$

where $WH_{\max}(f_\beta, w)$ is the minimum among all maximum absolute values of WHT over the $2^m - 1$ linear combinations of the output values of the S-box.

Definition 2.10. *Autocorrelation of S-box*: The autocorrelation of an S-box is denoted as $\hat{r}_\beta(s)$, where $s \in \mathbf{B}^n \setminus \{0^n\}$, and $\beta \in \mathbf{B}^m \setminus \{0^m\}$, and it is the highest autocorrelation value over the $2^m - 1$ linear combinations of the output values of the S-box.

Definition 2.11. *Balanced S-box*: An S-box is called balanced if the number of preimages of each output is 2^{n-m}.

Definition 2.12. *Bijective S-box*: An S-box is called bijective if $n = m$ and the number of preimages of each output is 1. Note that a bijective S-box is also balanced. Bijective S-boxes are also denoted as $n \times n$ S-boxes.

The quality of an S-box is primarily assessed by its properties that render it invulnerable to common attacks such as linear [5] and differential cryptanalysis [6]. In this context, nonlinearity and autocorrelation are essential properties. S-boxes of *high nonlinearity* and *low autocorrelation* become less vulnerable to attacks. Consequently, these two properties are typically adopted as quality measures of S-boxes and, hence, they are at the center of interest in the underlying optimization problems.

In order to maximize nonlinearity, the definition of Eq. (2) can be used. Equivalently, the maximization problem is transformed into a minimization problem over all possible S-boxes f, as follows:

$$\min_f \max_{\substack{\beta \in \mathbf{B}^m \\ w \in \mathbf{B}^n}} \left|\hat{F}_\beta(w)\right|, \tag{3}$$

where $\hat{F}_\beta(w)$ is the WHT defined in Eq. (1). Regarding autocorrelation, the minimization problem is defined as:

$$\min_f \max_{\substack{\beta \in \mathbf{B}^m \setminus \{0^m\} \\ s \in \mathbf{B}^n \setminus \{0^n\}}} \left|\hat{r}_\beta(s)\right|, \tag{4}$$

where $\hat{r}_\beta(s)$ is the autocorrelation defined in Definition 2.10.

The minimization problems of Eqs. (3) and (4) are consistently considered in the relevant literature for the design of S-boxes of high nonlinearity and low autocorrelation [10–12]. Typically, the nonlinearity is considered as the primary optimization objective, while autocorrelation is mostly reported for the final solution. Probably, this is due to the computational overhead imposed by the concurrent handling of both objectives.

3. Employed algorithms

In the following paragraphs, we outline the considered sequential algorithms as well as the proposed parallel APs. For presentation simplicity, we henceforth assume that the considered minimization problem is given in the general form

$$\min_{x \in \mathbf{X}} \phi(x),$$

where $\mathbf{X}$ in the corresponding search space.

3.1. Tabu Search

Tabu Search (TS) is a well-studied metaheuristic algorithm for discrete optimization problems [24]. TS belongs to the class of trajectory-based search methods. Thus, starting from a random initial position, it iteratively moves to the best position in the neighborhood of the current one. In order to overcome local minima, TS performs hill-climbing by changing from descending to ascending moves whenever a local minimum is detected.

In order to avoid cyclic moves, TS is equipped with a short-term memory, called the *tabu list* (TL), where recently visited positions are stored. The positions included in TL are prohibited for a number of iterations, thereby preventing the algorithm from retracing the same trajectories. The size of TL can affect the algorithm's performance. Satisfactory values for TL are typically problem-dependent.

Moreover, in order to avoid restraining the search in narrow parts of the search space, TS is usually applied within a multi-start framework. Thus, the best position of the current trajectory is monitored and, if it is not improved for a number of iterations, the algorithm is restarted on a new (randomly selected) initial position. Eventually, the algorithm terminates when it exceeds a predefined computational budget or the best found position has not been improved for a maximum number of restarts.

Let $x^{(t)} \in \mathbf{X}$ denote the current position of the algorithm at iteration t, and $N_x^{(t)} \subset \mathbf{X}$ be its neighborhood. Also, let $TL^{(t)}$ be the TL at iteration t, and $\tilde{N}_x{}^{(t)} \subset N_x^{(t)}$ contain all elements of $N_x^{(t)}$ that are not included in TL. Then, if x^* is the best detected position, the main trajectory-generation procedure of TS is outlined in Algorithm 1. A detailed analysis of the TS algorithm can be found in [26].

Algorithm 1. Pseudocode of the TS algorithm.

```
1:  procedure TS
2:    t ⟵ 0
3:    while (not termination) do
4:      x^(t+1) ⟵ argmin_{y ∈ Ñ_x^(t)} f(y)
5:      TL^(t+1) ⟵ TL^(t) ∪ {x^(t+1)} and remove the oldest entry from TL^(t+1)
6:      if (φ(x^(t+1)) < φ(x*)) then
7:        x* ⟵ x^(t+1)
8:      end if
9:      t ⟵ t + 1
10:   end while
11: end procedure
```

Algorithm 2. Pseudocode of the SA algorithm.

```
1:  procedure SA
2:    t ⟵ 0, x ⟵ x^(t), T ⟵ T^(t)
3:    while (not termination) do
4:      for i = 1 ... S_T do
5:        Randomly select y ∈ N_x^(t) ∩ X
6:        ε ⟵ φ(y) − φ(x^(t))
7:        if (ε < 0) OR (rand() < exp(−ε/T)) then
8:          x^(t) ⟵ y
9:        end if
10:     end for
11:     T^(t+1) ⟵ α T^(t)
12:     t ⟵ t + 1
13:   end while
14: end procedure
```

3.2. Simulated Annealing

Simulated Annealing (SA) is a popular trajectory-based metaheuristic, particularly used in discrete optimization problems [25]. SA is equipped with a hill-climbing mechanism, which is based on the probabilistic acceptance of non-improving solutions in order to alleviate local minimizers. Let $x^{(t)} \in \mathbf{X}$ be the current position at iteration t, and $N_x^{(t)} \subset \mathbf{X}$ be its neighborhood. Also, let $T^{(t)}$ be a parameter, called the *temperature*, which controls the probability of accepting non-improving solutions. The algorithm starts from a random initial position, $x^{(0)}$, and a (usually high) initial temperature $T^{(0)}$. At each iteration, SA performs a number, S_T, of inner steps with fixed temperature value. Then, the temperature is scaled according to a user-defined *cooling factor*, $\alpha \in (0, 1)$, and the algorithm proceeds to the next iteration.

At each inner step, a point y is randomly and uniformly selected from $N_x^{(t)}$. In case of improvement (better objective function value), the algorithm moves to the new position y. If y is a non-improving position, the algorithm accepts it with probability $p = \exp(-\varepsilon/T)$, where $\varepsilon = f(y) - f\left(x^{(t)}\right)$. Note that the probability p increases with the value of T. This promotes more frequent acceptance of non-improving solutions at early stages of the optimization process, thereby enhancing the exploration capability of the algorithm. As T decreases, the algorithm tends to accept only improving solutions, amplifying its exploitation dynamic.

SA terminates its execution when a predefined computational budget is exceeded or if it fails to improve the best solution for a number of iterations. The main procedure of the SA considered in the present work is given in Algorithm 2, where *rand*() stands for a uniform pseudorandom numbers generator in the range [0, 1]. For a detailed presentation of SA, the reader is referred to [27].

3.3. Algorithm Portfolios

Algorithm Portfolios (APs) define a general framework where multiple algorithms or instances of the same algorithm are executed either simultaneously (in parallel environments) or interchangeably (in single-processor machines) [13,14]. The rationale behind APs lies in the fact that the best algorithm for a given problem is rarely *a priori* known. Thus, allocating the available computational budget to more, preferably diverse, algorithms can increase the probability of finding good solutions, while reducing the *risk* in terms of deviation of the final solution in multiple experiments.

Algorithm 3. Algorithm Portfolios: Pseudocode of master node.

```
1:  procedure MASTER()
2:    Initialize N slave-nodes and assign an algorithm to each one
3:    t ⟵ 0, φ_best ⟵ ∞
4:    while (nodes still running) do
5:      Receive x_[i], φ_[i] = φ(x_[i]) from node i
6:      if (φ_[i] < φ_best) then
7:        φ_best ⟵ φ_[i], x_best ⟵ x_[i]
8:      end if
9:    end while
10:   Report x_best
11: end procedure
```

Algorithm 4. Algorithm Portfolios: Pseudocode of slave node.

```
1:  procedure SLAVE()
2:    Initialize assigned algorithm
3:    while (allocated time not exceeded) do
4:      execute one iteration of the algorithm
5:      if (new best x_[i] found) then
6:        Send x_[i], φ_[i] = φ(x_[i]) to master node
7:      end if
8:    end while
9:    Terminate slave node
10: end procedure
```

APs have recently gained increasing popularity [15–22]. They have been applied both in serial and parallel environments for various problems of one or many objective functions. Parallel APs have been proposed to tackle demanding optimization problems, such as the problems emanating from Operations Research [19,28,29] and Combinatorics [30]. Simple parallelization models have been employed, where each algorithm of the AP is allocated to a single processor. In [19,30] the proposed APs are based on a master-slave parallelization model and exchange information via a message passing protocol. The role of the master node in the model is to distribute the available execution time budget among the algorithms according to a sophisticated trading-based mechanism. On the other hand, the slave nodes are devoted solely to the execution of the optimization algorithms.

The algorithms that comprise an AP can be either isolated (*non-interactive AP*) or communicate and exchange information with each other (*interactive AP*). The communicated information usually comprises the best solutions detected by each algorithm. This model is typically used in APs of population-based algorithms. For example, in [21] the proposed AP accommodates a number of population-based algorithms that exchange solutions via regular migrations. In [22] the constituent population-based algorithms of the AP interact with each other by exchanging information via a shared population of search points. Alternatively, the algorithms of the AP can communicate by trading elite solutions at the cost of execution time, as in the models proposed in [19,30].

Non-interactive APs have also been applied on problems that originate from Operations Research [28,29] and Combinatorial Optimization [31]. It has been shown that the lack of information exchange can be beneficial in some applications where the search can rapidly become biased in narrow neighborhoods of the search space. In such cases, the infusion of solutions between the algorithms can eventually suppress their exploration capability or lead to premature convergence [16,17].

The dilemma of allowing information exchange or not shall take into consideration the specific algorithms used in the AP. For example, an AP that includes both population-based and trajectory-based algorithms may gain more benefits from information exchange. This is because the trajectory-based algorithm has the capability of local search, which complements the inher-

ent global search properties of the population-based algorithm. Non-interactive APs are habitually considered first in applications, followed by their interactive variants.

The present work offers a first study of APs on the bijective S-boxes design problem. We consider non-interactive APs consisting of the state-of-the-art TS and SA algorithms running in parallel according to a master-slave model. Each constituent algorithm runs individually on a single slave node without information exchange. Whenever an algorithm detects a new solution that improves its current one, it sends it to the master node. The master node is responsible for all book-keeping operations, storing the best solutions discovered by the algorithms. Pseudocode of the proposed APs is provided in Algorithm 3 (master node) and Algorithm 4 (slave nodes). Further implementation and parameterization details are provided in the following paragraphs.

3.4. Implementation details

For the implementation of the considered algorithms, first we focus on the representation of the solution vector. We aim at detecting bijective $n \times n$ S-boxes with n binary inputs and n binary outputs. The number of combinations of all binary inputs is equal to 2^n, and each binary input vector has an n-bit output vector. Therefore, the S-box can be represented as a binary solution vector of dimension

$$D_{\text{bin}} = 2^n \times n.$$

Evidently, the S-boxes design problem is equivalent to a high-dimensional binary optimization problem.

Moreover, the studied S-boxes are bijective as well as balanced. Therefore, each one of the 2^n binary output vectors is mapped to only one of the 2^n binary input vectors. Taking into consideration this property, we adopt the solution-generation technique reported in [32]. This technique constructs all the 2^n possible output vectors of the S-box. Then a candidate solution (S-box) is formed by assigning each n-bit output vector to only one of the 2^n input vectors. Initially the assignment is randomly and uniformly performed as in [10,11].

Using this technique, the optimization algorithms need to find the proper mapping between the 2^n binary input vectors and the 2^n binary output vectors, i.e., the one that maximizes nonlinearity and minimizes autocorrelation of the resulting S-box. This is achieved by searching for the best position permutation of the 2^n binary output vectors in the S-box. Thus, the D_{bin}-dimensional binary optimization problem is transformed into a D-permutation problem where

$$D = 2^n. \tag{5}$$

This technique retains vectors that satisfy the requirement for equal number of 0 and 1 in the solution, and it has been successfully used with Cartesian Genetic Programming [32]. Note that the resulting search spaces are vast even for small values of n, since their cardinality is equal to the factorial $(2^n)!$.

The evaluation of the generated candidate solutions is based on both nonlinearity and autocorrelation although with different priority. Specifically, higher priority is given to nonlinearity and lower priority to autocorrelation. This is in accordance with the reported significance of the two objectives in relevant literature [33,34]. It results in a two-stage solution assessment scheme that imitates multi-objective optimization, consisting of the following rules:

- Between two candidate solutions, the one that achieves the highest nonlinearity is preferable.
- Between two candidate solutions of equal nonlinearity, the one that has the lowest autocorrelation is preferable.

The algorithms are compared on the basis of their final solution quality, i.e., its nonlinearity and autocorrelation, according to these rules. In case of equivalent solutions in both nonlinearity and autocorrelation, the required running times of the algorithms to achieve their final solutions serve as the tiebreaker.

Table 1
Problem size, dimension D of permutation vector, cardinality $|\mathbf{X}|$ of search space, and available time budget (in minutes) per experiment.

Problem	D	$\lvert\mathbf{X}\rvert$	Time (min)
5×5	32	$\sim 10^{35}$	2
6×6	64	$\sim 10^{89}$	60
7×7	128	$\sim 10^{215}$	480
8×8	256	$\sim 10^{506}$	1440

4. Experimental evaluation

We considered the sequential TS and SA algorithms as the baseline approaches for comparisons. Note that SA has been previously used on S-boxes design problems [12,35] while, to the best of our knowledge, TS is used for the first time. We also considered various instances of the proposed parallel APs with different constituent TS and SA variants. Each AP followed a master-slave model and initializes a number of nodes. These can be either individual processors or threads in a multi-core processor. In our experiments the APs were run on 3 and 5 nodes. Henceforth, we use the following notation:

- *TS*: sequential TS algorithm.
- *SA*: sequential SA algorithm.
- *AP*(*TS*, *k*): homogeneous AP on *k* CPUs, comprising solely TS instances.
- *AP*(*SA*, *k*): homogeneous AP on *k* CPUs, comprising solely SA instances.
- *AP*(*TS*, *SA*, *k*): heterogeneous AP on *k* CPUs, comprising both TS and SA instances.

In all parallel APs, the master node served only for book-keeping and solution storage purposes, while the slave nodes were devoted to the algorithms. Thus, the APs on 3 and 5 nodes contained 2 and 4 algorithms, respectively.

The experimental evaluation was conducted on the `saw` cluster of the WestGrid[1] consortium. The APs were run on Intel© Xeon 2.83 GHz processors with 16 GB RAM. All source codes were developed in the `C` programming language. For the parallelization, the OpenMPI project[2] libraries were used with the `gcc` compiler.

Our test suite included four bijective S-boxes design problems that have been commonly used for benchmarking in relevant works. In parallel computing environments, the running time of the algorithms has special merit [36]. Usually, strict restrictions apply on the maximum available time per user, while significant costs may apply to users occupying the machines for long time. Adhering to these requirements, we considered running time to be the computational budget in our experimental study. Different budget was provided for each test problem, taking its complexity into consideration. The considered test problems, the dimension D of the corresponding permutation vectors, the cardinality of the search space, and the corresponding time budget per experiment are reported in Table 1. The imposed time limits are challenging

[1] http://www.westgrid.ca.
[2] http://www.open-mpi.org.

Table 2
Parameter settings of the algorithms.

Algorithm	Parameter	Description	Value(s)
AP	N	Number of nodes	3, 5
TS	LS	Tabu list size	D
	T_{noimp}	Non-improving iterations before restart	100
SA	S_T	Number of inner steps	1500, 2500
	$T^{(0)}$	Initial temperature	1.0
	α	Cooling factor	0.5, 0.98
	T_{noimp}	Non-improving iterations before restart	50

since, in similar works, running times that span even several days have been reported.

The execution of each algorithm was terminated as soon as it exceeded the predefined running time. In the parallel APs, the total running time was equally allocated to the constituent algorithms. A number of 25 independent experiments was conducted per algorithm and test problem. At each experiment, the best detected solution (S-box) and the elapsed running time until its detection was recorded.

In statistical comparisons, the algorithms were assessed on the basis of solution quality, i.e., the nonlinearity (primarily) and autocorrelation (secondarily) of the achieved solutions, as analyzed in Section 3.4. In case of ties under these criteria, the required running times for attaining the solutions were used to identify the dominant algorithm.

4.1. Sequential algorithms

The first round of experiments was devoted to the sequential versions of TS and SA. It is well established that parameterization affects the algorithms' performance. For this reason, we considered the following two settings for both algorithms:

- Fixed parameter values according to trial-and-error preprocessing.
- Randomly assigned parameters.

The obtained parameter values for the first case are reported in Table 2. Note that the parameters S_T and α of SA exhibited two different promising values each, resulting in four distinct SA instances. We will henceforth use the following notation for the sequential approaches:

(1) TS: TS with fixed parameter $LS = D$.
(2) TS_r: TS with randomly and uniformly selected LS in the range $[D, 2D]$.
(3) SA_1: SA with $S_T = 1500$ and $\alpha = 0.5$.
(4) SA_2: SA with $S_T = 1500$ and $\alpha = 0.98$.
(5) SA_3: SA with $S_T = 2500$ and $\alpha = 0.5$.
(6) SA_4: SA with $S_T = 2500$ and $\alpha = 0.98$.
(7) SA_r: SA with randomly and uniformly selected S_T and α in the ranges [1500, 2500] and [0.5, 0.98], respectively.

For each test problem and algorithm, the obtained solutions over the 25 experiments were ranked according to their quality, i.e., their nonlinearity (NL) value (primarily), their autocorrelation (AC) value (secondarily), and the required running time otherwise. The median and the best solutions in the ranking are reported for the sequential TS and SA algorithms in Tables 3 and 4, respectively. The median was preferred against the mean due to the discrete (integer) nature of the NL and AC objectives, as well as due to its use in the considered statistical significance tests.

As reported in Table 3, the two TS approaches achieved identical median solutions in terms of NL and AC for the 5×5 and 6×6 problems. In the rest of the problems, the AC values achieved by TS were better than that of TS_r. This is reasonable since TS admitted carefully selected parameters through preprocessing. The best solutions were identical in all cases. For the most challenging 7×7 and 8×8 problems, TS_r achieved a solution of inferior AC quality but same NL value with TS. On the other hand, as the problem dimension increases, TS_r seems to require shorter running time than TS to attain solutions of equal quality with the reported medians.

Our experimentation was supported by pairwise Wilcoxon rank-sum tests at significance level 95% among the algorithms. For each test problem, the solutions provided by TS were compared against those of TS_r with respect to NL (primarily), AC (secondarily), and running time otherwise. In case of statistically significant differences between the two algorithms, the dominating one was awarded a *win*, while a *loss* was counted for the other. In case of statistical indifference, both algorithms were awarded a *draw*. The conducted tests revealed insignificant differences in almost all test problems, verifying the mild parameter sensitivity of the TS algorithm on the considered problems. The exception to this was the 7×7 problem, where TS was the dominant algorithm in terms of wins. This is marked with the star "*" symbol in Table 3.

The corresponding results for the SA algorithms are reported in Table 4. Although there are only few differences in the reported median and best values for the NL, considerable differences are observed for the AC values. A closer look in Table 4 reveals that the SA_1 and SA_3 variants, which assumed the $\alpha = 0.5$ parameter, exhibited superior performance, always attaining the best solution. This indicates that rapid modulation of the temperature parameter can be beneficial for the specific problems under our strict computational budgets. The only exception is the 6×6 problems, where the SA_2 variant discovered the overall best solution in terms of NL. However, even in this case, the AC of this solution was higher than the aforementioned approaches. Interestingly, the random-parameter variant SA_r closely followed in performance the fixed-parameter variants.

Table 3
Results for the sequential TS algorithms.

Problem	Alg.		Median			Best		
$n \times m$			NL	AC	Time (ms)	NL	AC	Time (ms)
5×5	TS		10	16	15,184.24	10	16	1492.77
	TS_r		10	16	7603.43	10	16	1564.04
6×6	TS		20	32	7936.34	20	32	3365.81
	TS_r		20	32	7342.95	20	32	3502.38
7×7	TS	*	46	48	7,332,689.32	46	48	4,540,495.85
	TS_r		46	56	2,870,727.66	46	48	2,242,817.24
8×8	TS		98	80	78,001,050.97	100	80	61,694,764.99
	TS_r		98	88	9,287,593.35	100	80	83,956,598.72

Table 4
Results for the sequential SA algorithms.

Problem	Alg.		Median			Best		
$n \times m$			NL	AC	Time (ms)	NL	AC	Time (ms)
5×5	SA_1	*	10	16	23,985.54	10	16	11,168.44
	SA_2		10	16	48,594.72	10	16	37,122.27
	SA_3	*	10	16	9057.50	10	16	4995.97
	SA_4		10	24	12,210.65	10	16	35,346.25
	SA_r	*	10	16	44,607.71	10	16	22,363.94
6×6	SA_1	*	20	32	2568.39	20	32	2379.13
	SA_2		20	32	34,616.83	22	40	814,711.13
	SA_3	*	20	32	3983.65	20	32	3961.48
	SA_4		20	32	67,908.50	20	32	3951.24
	SA_r		20	32	13,223.55	20	32	2853.44
7×7	SA_1	*	46	48	28,461,662.76	46	48	1,375,571.42
	SA_2		46	56	243,804.27	46	56	51,618.37
	SA_3	*	46	48	15,404,001.48	46	48	1,639,139.68
	SA_4		46	56	405,697.99	46	56	142,187.42
	SA_r		46	56	192,213.00	46	48	4,992,287.41
8×8	SA_1	*	100	88	2,827,104.23	100	80	1,769,426.69
	SA_2		100	88	30,456,410.60	100	88	1,019,285.79
	SA_3	*	100	88	2,644,293.84	100	80	2,230,341.29
	SA_4		100	88	37,945,562.16	100	88	1,890,570.36
	SA_r	*	100	88	9,521,700.16	100	80	10,031,840.50

Pairwise Wilcoxon rank-sum tests were conducted among all SA variants for each test problem. For each test problem, the dominant approaches in terms of wins are marked with a star "*" symbol in Table 4, and they have statistically indifferent performance among them. Overall, the SA_3 variant is the most prominent as the problem dimension increases. Also, it achieved very competitive running times, especially for its best solutions. This can be attributed to the $S_T = 2500$ setting that, combined with the rapid decrease of the temperature due to $\alpha = 0.5$, promoted the rejection of non-improving moves more rapidly than in the rest of the SA approaches. Thus, it resulted in fast transition of the algorithm's dynamic from global to local search. Overall, SA appeared to be more sensitive on its parameter setting than TS, while its performance was highly competitive to TS.

4.2. Algorithm Portfolios

The second round of experiments focused on the proposed APs. Both fixed and random parameters were considered for the constituent TS and SA algorithms. Due to the large number of combinations, we restricted our experiments on APs comprising two variants of each algorithm. Thus, the promising *TS*, TS_r, SA_3, and SA_r algorithms were selected to form APs running on 3 and 5 nodes. The designed APs consisted of either a sole algorithm (homogeneous APs) or two different algorithms (heterogeneous APs).

For each test problem, 25 independent experiments were conducted per AP and they were statistically analyzed similarly to the sequential approaches. The obtained results are reported in Tables 5 and 6. Note that all the received solution values lie within the theoretical bounds reported in [9]. For each test problem, the APs are reported in blocks according to their constituent algorithm. The results of the baseline sequential algorithms, namely *TS*, TS_r, SA_3, and SA_r, are reproduced from Tables 3 and 4 to facilitate comparisons, and they are highlighted with light gray color in Tables 5 and 6.

For example, in the upper half of Table 5 (5×5 problem), the first block consists of the sequential *TS* algorithm and the two homogeneous TS-based APs, namely *AP*(*TS*, 3) and *AP*(*TS*, 5), with 3 and 5 nodes, respectively. Next comes the block of TS_r and its related homogeneous APs, followed by the blocks of SA_3 and SA_r, along with the corresponding homogeneous APs. The last block for

Table 5
Results of the APs for the 5×5 and 6×6 test problems.

Algorithm		Median			Best		
		NL	AC	Time (msec)	NL	AC	Time (msec)
5×5 S-box							
TS		10	16	15184.24	10	16	1492.77
$AP(TS, 3)$	*	10	16	6576.10	10	16	1039.90
$AP(TS, 5)$	* (2)	10	16	4080.50	10	16	643.20
TS_r		10	16	7603.43	10	16	1564.04
$AP(TS_r, 3)$		10	16	4430.70	10	16	814.80
$AP(TS_r, 5)$	* (1)	10	16	3651.60	10	16	1272.50
SA_3		10	16	9057.50	10	16	4995.97
$AP(SA_3, 3)$		10	16	3988.60	10	16	3988.60
$AP(SA_3, 5)$		10	16	10040.00	10	16	6645.80
SA_r		10	16	44607.71	10	16	22363.94
$AP(SA_r, 3)$	*	10	16	6479.80	10	16	4654.60
$AP(SA_r, 5)$	*	10	16	6790.80	10	16	2176.20
$AP(TS, SA_3, 3)$		10	16	10362.30	10	16	557.20
$AP(TS, SA_3, 5)$		10	16	4998.70	10	16	1552.90
$AP(TS_r, SA_r, 3)$		10	16	6777.50	10	16	1481.50
$AP(TS_r, SA_r, 5)$	*	10	16	4555.60	10	16	1020.30
6×6 S-box							
TS		20	32	7936.34	20	32	3365.81
$AP(TS, 3)$		20	32	6803.30	22	32	657451.30
$AP(TS, 5)$	*	20	32	6693.30	20	32	3355.80
TS_r		20	32	7342.95	20	32	3502.38
$AP(TS_r, 3)$		20	32	7455.60	22	32	1193029.70
$AP(TS_r, 5)$	*	20	32	3904.10	22	32	881951.40
SA_3	* (1)	20	32	3983.65	20	32	3961.48
$AP(SA_3, 3)$	(2)	20	32	4026.40	20	32	3973.60
$AP(SA_3, 5)$		20	32	4130.80	20	32	4052.50
SA_r		20	32	13223.55	20	32	2853.44
$AP(SA_r, 3)$		20	32	3807.80	20	32	2519.80
$AP(SA_r, 5)$	*	20	32	3799.30	20	32	2542.20
$AP(TS, SA_3, 3)$		20	32	4390.40	20	32	3844.70
$AP(TS, SA_3, 5)$	*	20	32	4224.80	20	32	3585.90
$AP(TS_r, SA_r, 3)$		20	32	4185.40	20	32	3031.50
$AP(TS_r, SA_r, 5)$		20	32	4136.00	20	32	2762.60

Table 6
Results of the APs for the 7×7 and 8×8 test problems.

Algorithm		Median			Best		
		NL	AC	Time (msec)	NL	AC	Time (msec)
7×7 S-box							
TS	$*^{(1)}$	46	48	7332689.32	46	48	4540495.85
$AP(TS, 3)$		46	48	10670926.70	46	48	531649.50
$AP(TS, 5)$	*	46	56	404443.60	46	48	1725213.90
TS_r	*	46	56	2870727.66	46	48	2242817.24
$AP(TS_r, 3)$	*	46	48	6859680.70	46	48	640373.80
$AP(TS_r, 5)$	*	46	56	960809.10	46	48	4461868.70
SA_3		46	48	15404001.48	46	48	1639139.68
$AP(SA_3, 3)$	*	46	48	12653260.80	46	48	276356.10
$AP(SA_3, 5)$	$*^{(1)}$	46	48	3155517.80	46	48	183949.40
SA_r		46	56	192213.00	46	48	4992287.41
$AP(SA_r, 3)$	*	46	56	33075.10	46	48	2417257.80
$AP(SA_r, 5)$	*	46	56	114760.80	46	48	624332.80
$AP(TS, SA_3, 3)$	*	46	56	102293.00	46	48	375019.10
$AP(TS, SA_3, 5)$	*	46	48	6029739.40	46	48	847927.20
$AP(TS_r, SA_r, 3)$	*	46	56	110495.10	46	48	5592137.80
$AP(TS_r, SA_r, 5)$		46	56	159311.90	46	48	3156178.30
8×8 S-box							
TS	*	98	80	78001050.97	100	80	61694764.99
$AP(TS, 3)$		98	88	11125639.50	98	80	18167071.70
$AP(TS, 5)$		98	88	13297265.00	98	80	15577934.10
TS_r		98	88	9287593.35	100	80	83956598.72
$AP(TS_r, 3)$	*	98	88	9266743.50	98	80	15573342.40
$AP(TS_r, 5)$		98	88	10818025.60	98	80	9497480.20
SA_3	$^{(1)}$	100	88	2644293.84	100	80	2230341.29
$AP(SA_3, 3)$	*	100	88	910879.20	100	80	5821906.80
$AP(SA_3, 5)$	$^{(2)}$	100	80	16348693.20	100	80	3554649.30
SA_r		100	88	9521700.16	100	80	10031840.50
$AP(SA_r, 3)$		100	88	5256191.20	100	80	19673499.00
$AP(SA_r, 5)$		100	88	7079210.10	100	80	4949492.80
$AP(TS, SA_3, 3)$	*	100	88	4023823.20	100	80	10780589.40
$AP(TS, SA_3, 5)$	*	100	88	2910635.10	100	80	2351611.20
$AP(TS_r, SA_r, 3)$		100	88	15498679.60	100	88	8189413.10
$AP(TS_r, SA_r, 5)$		100	88	5374450.40	100	80	5179593.80

the 5×5 problem reports results for the heterogeneous APs that combine fixed- and random-parameter variants of the constituent algorithms.

The dominating approaches in terms of wins in the pairwise Wilcoxon rank-sum tests are marked with a star "*" symbol in Tables 5 and 6. In addition, all approaches competed against each other and the two best-performing algorithms in terms of wins in the rank-sum tests are marked with the corresponding superscripts. For example, for the 5×5 problem in Table 5, $AP(TS_r, 5)$ was ranked first, while $AP(TS, 5)$ was ranked second among the best-performing approaches. Similarly, SA_3 and $AP(SA_3, 3)$ were the best algorithms for the 6×6 problem. Note that starred algorithms within the same block had equal numbers of wins among them. The total number of wins per algorithm and test problem are also graphically illustrated in Fig. 1.

Diverse conclusions can be derived from the reported results. In the 5×5 problem, homogeneous APs appear to be superior to their sequential counterparts in terms of the reported median solution values or time efficiency (in case of equal quality). The heterogeneous APs appear to consistently achieve high-quality median solutions and high time-efficiency compared to the baseline algorithms. Similar conclusions are derived for the reported best solutions.

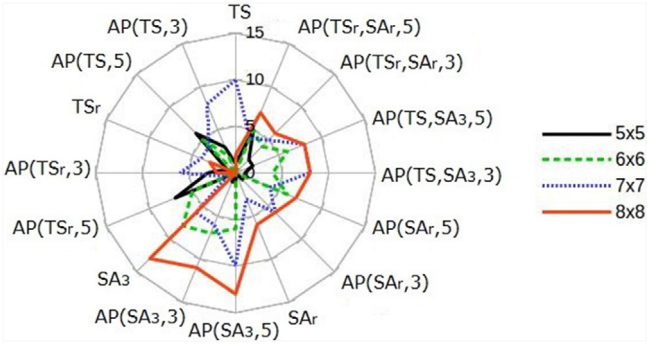
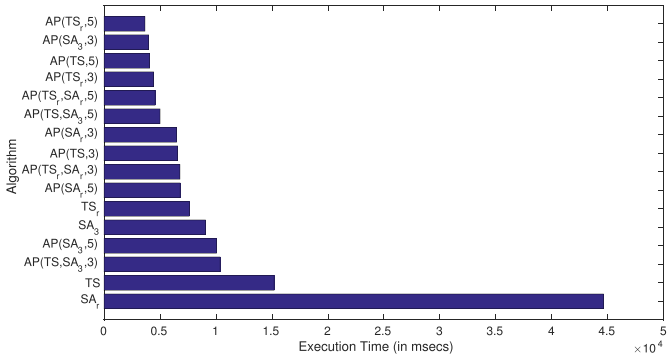
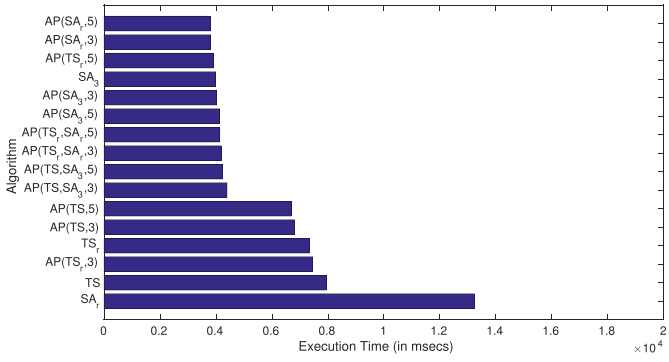


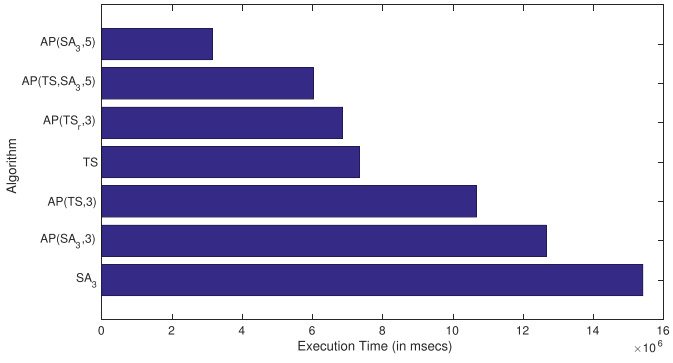
Fig. 1. Number of wins per algorithm and test problem.



(a) 5×5 test problem.

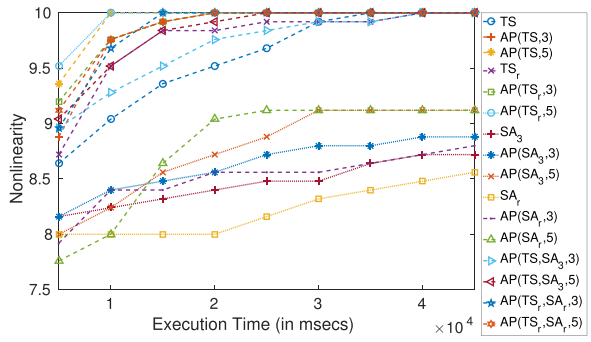


(b) 6×6 test problem.

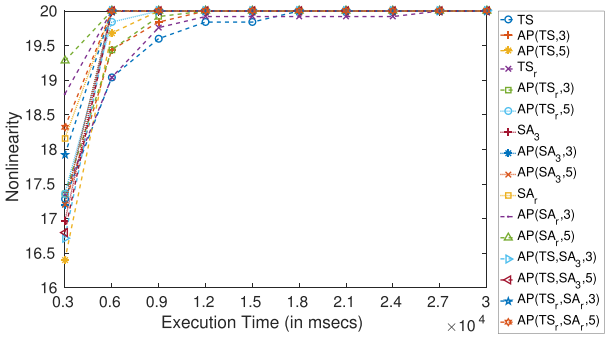


(c) 7×7 test problem.

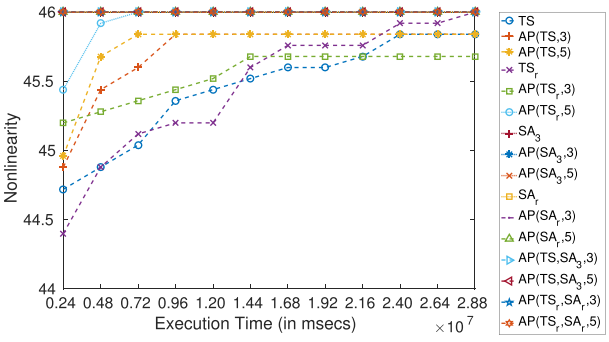
Fig. 2. Required execution time to achieve the reported median solutions.



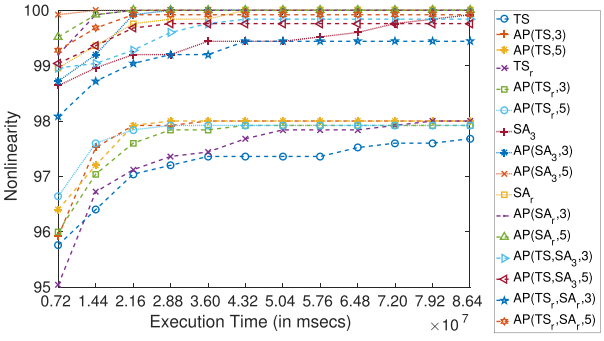
(a) 5×5 test problem.



(b) 6×6 test problem.

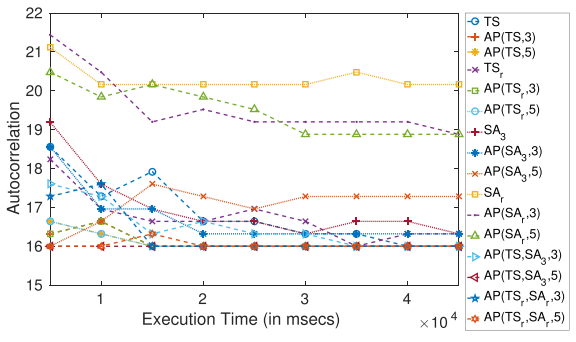


(c) 7×7 test problem.

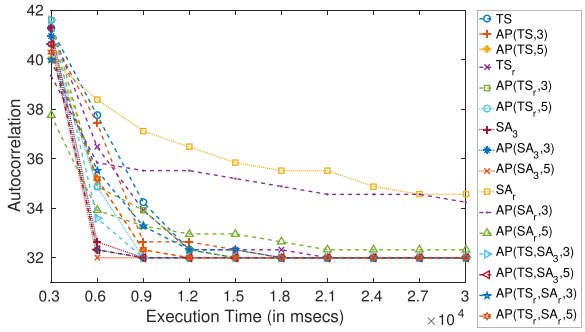


(d) 8×8 test problem.

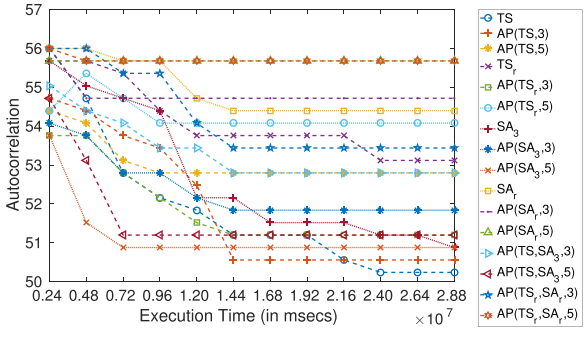
Fig. 3. Nonlinearity of the best solution during the algorithms' execution.



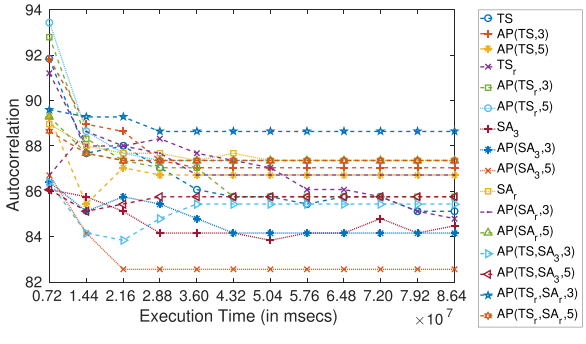
(a) 5×5 test problem.



(b) 6×6 test problem.



(c) 7×7 test problem.



(d) 8×8 test problem.

Fig. 4. Autocorrelation of the best solution during the algorithms' execution.

Another interesting observation is that APs with 5 nodes outperformed those with 3 nodes. This is an empirical confirmation of the rationale behind the use of multiple algorithms in APs. Moreover, the overall best AP in our experiments was the $AP(TS_r, 5)$, which consists of four instances of the TS algorithm with random parameters, followed by the homogeneous $AP(TS, 5)$ that consists of four fixed-parameter instances of TS. Similar performance was attained for the 6×6 problem. The APs habitually outperformed the sequential algorithms. However, in this case the SA_3 approach was ranked first, followed by $AP(SA_3, 5)$. Clearly, the previously neglected SA_3 algorithm appeared to form efficient AP schemes in this case.

Table 7
Median numbers of function evaluations (FEV) required by the APs for all test problems.

Algorithm	5×5			6×6			7×7			8×8		
	NL	AC	FEV	NL	AC	FEV	NL	AC	FEV	NL	AC	FEV
$AP(TS, 3)$	10	16	27,698	20	32	2172	46	48	641,354	98	88	65,297
$AP(TS, 5)$	10	16	16,857	20	32	2030	46	56	16,516	98	88	66,930
$AP(TS_r, 3)$	10	16	19,026	20	32	2112	46	48	471,266	98	88	49,958
$AP(TS_r, 5)$	10	16	14,583	20	32	2006	46	56	40,686	98	88	35,425
$AP(SA_3, 3)$	10	16	16,106	20	32	1090	46	48	878,650	100	88	6032
$AP(SA_3, 5)$	10	16	43,412	20	32	1048	46	48	237,382	100	80	111,113
$AP(SA_r, 3)$	10	16	28,772	20	32	2375	46	56	1743	100	88	54,970
$AP(SA_r, 5)$	10	16	30,124	20	32	1575	46	56	7684	100	88	57,190
$AP(TS, SA_3, 3)$	10	16	43,742	20	32	1074	46	56	6019	100	88	36,013
$AP(TS, SA_3, 5)$	10	16	21,094	20	32	1024	46	48	351,183	100	88	26,208
$AP(TS_r, SA_r, 3)$	10	16	28,423	20	32	2039	46	56	7626	100	88	131,924
$AP(TS_r, SA_r, 5)$	10	16	19,769	20	32	2010	46	56	11,140	100	88	58,931

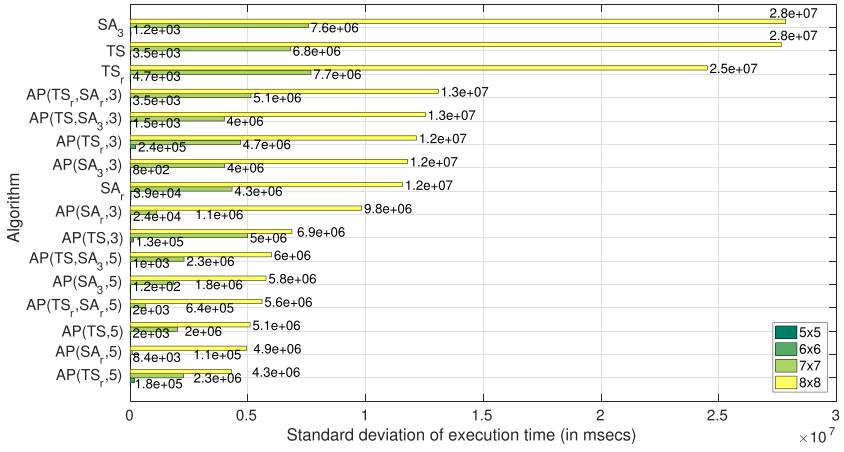


Fig. 5. Standard deviation of running time for all algorithms and problems.

In the 7×7 and 8×8 problems reported in Table 6, enhanced performance was achieved for the TS-based APs with 3 nodes. This can be attributed to the exponentially increasing difficulty of the problem as the size of the S-box increases linearly, which augments the time requirements of the algorithms for the detection of high-quality solutions. The constituent algorithms of the APs with 3 nodes were assigned more time than the algorithms in APs with 5 nodes (recall that the total computational budget is the same for all APs). Thus, they could search the search space more thoroughly, providing better solutions. This is a strong indication that the benefits from the use of APs decline if the running time per constituent algorithm is inadequate for deploying its dynamics.

In contrast to TS-based APs, the SA-based APs with 5 nodes outperformed the ones with the 3 nodes in most test problems. This suggests that SA-based APs exploited the additional nodes more efficiently. In particular, the inherent randomization of *SA* proved to enhance the exploration dynamics of the APs, which is beneficial in challenging problems. In fact, for the 8×8 test problem, $AP(SA_3, 5)$ was the best AP with marginal difference from the sequential SA_3 approach, which was ranked first mostly due to its rapid detection of the best solutions. This verifies the necessity of providing adequate running time to the algorithms when dimension increases. Note that $AP(SA_3, 5)$ was the variant that achieved the best median solution as we can see in Table 6.

Overall, the considered APs exhibited competitive performance in all test problems. They were able to regularly achieve high-quality solutions and time efficiency. This is clearly communicated from Fig. 2, where the algorithms that achieved the best medians per problem are ranked according to their running times. In all cases, APs occupy higher positions due to their high time-efficiency. The 8×8 test problem is excluded because only the $AP(SA_3, 5)$ variant achieved the best median solution. Another interesting observation is that APs based on SA_3 appear to be the most efficient. This implies that SA-based APs gain more benefits from fixed-parameter settings.

As previously mentioned, in parallel computing environments the running time of an algorithm constitutes an established measure of efficiency. However, for completeness purposes, the median of the number of function evaluations required by the APs for each test problem is reported in Table 7. In general, the observed performance with respect to function evaluations is aligned with the previous findings based on the running time of the algorithms.

For better understanding of the algorithms, the progress of their NL value during their execution, averaged over the 25 experiments, is depicted in Fig. 3. The corresponding AC values are illustrated in Fig. 4. Specifically, we divided the running time into equal segments and plotted the achieved average NL and AC per segment. The curves yield apparent NL differences between the sequential algorithms and the APs. Also, we can see that AC is not monotonically decreasing but rather fluctuates. This is anticipated since AC is a lower-priority objective compelled to follow the changes in NL.

Another issue of interest in APs is the reduction of risk, which is defined in terms of the standard deviation of the AP's time efficiency. For the employed algorithms this can be interpreted as the standard deviation of the running time. Fig. 5 reveals the standard deviations of running times for all algorithms and problems. Apparently APs achieved smaller standard deviations, especially for the harder problems, thereby revealing their risk-reduction properties.

Table 8
Maximum nonlinearity values achieved by the considered algorithms against other approaches in the literature.

Problem	Current work	Picek et al. [37]	Laskari et al. [10]	Clark et al. [12]	Millan et al. [38]
5×5	10	10	10	10	10
6×6	22	22	20	22	20
7×7	46	48	46	48	46
8×8	100	104	98	102	100

Table 9
Minimum autocorrelation values achieved by the considered algorithms against other approaches in the literature.

Problem	Current work	Laskari et al. [10]	Clark et al. [12]
5×5	16	16	16
6×6	32	32	32
7×7	48	56	48
8×8	80	80	80

Finally, for completeness purpose, Tables 8 and 9 report the best NL and AC values achieved by the proposed algorithms within the provided strict time budgets, against other computational methods from the literature. Specifically, the best results of Picek et al. [37], Laskari et al. [10], Clark et al. [12], and Millan et al. [38] are reported. Regarding the NL values, we witness some differences mainly for the most difficult problems. In particular, for the 7×7 and 8×8 problems, the cost functions proposed in [37,12] resulted in solutions of high NL values. The use of these cost functions constitutes an interesting direction for future research, as they can be incorporated in the proposed APs to boost their performance.

For the 8×8 test problem, some additional works can be found in the literature. In [39] the proposed approach was based on a finite field inversion method and managed to detect 8×8 S-boxes with NL value equal to 106. In [40], a reverse genetic algorithm with initial population of AES affine equivalent S-boxes was used, succeeding to detect solutions with NL value 112. Finally, in [41] the proposed approach exploited important theoretical findings on power mappings. The employed method managed to detect S-boxes with NL values equal to 112. Regarding autocorrelation, Table 9 shows that the other approaches achieved similar results to the studied APs. The main difference is observed in the 7×7 problem, where our approach achieved the best AC along with the approach of [12].

5. Conclusions

The design of bijective S-boxes of high nonlinearity and low autocorrelation is a hard computational optimization task. In the present work, we applied two well-studied trajectory methods, namely TS and SA, on the S-boxes design problem. To the best of our knowledge, TS has never been applied to design bijective S-boxes before. For the considered problem, we also proposed parallel APs based on the widely used master-slave model. The APs were constructed using either one algorithm (homogeneous case) or both TS and SA (heterogeneous case), and were thoroughly compared with each other as well as with their constituent algorithms. Extensive experimental evaluation revealed that the proposed APs can provide S-boxes of better or equal quality with the sequential algorithms, although in significantly less time. SA-based APs that used fixed-parameter settings usually outperformed TS-based ones in harder test problems while in smaller test problems, TS-based APs surmounted SA-based ones. Additionally, TS-based APs achieved smaller standard deviations of running time compared to SA-based ones. In general, the proposed APs achieved smaller deviations in the time required to attain their best solution compared to their sequential versions, especially for the harder problems. Future research will include the exploration of different AP models based on performance forecasting techniques, as well as interactive AP frameworks. Also, more sophisticated formulations of the problem will be explored, along with different cost functions such as the spectrum-based cost function reported in the relevant literature.

Acknowledgements

The authors would like to thank the anonymous reviewers for their critical comments and constructive criticism. Also, the authors would like to thank the Shared Hierarchical Academic Research Computing Network (SHARCNET), as well as the WestGrid HPC consortium for providing the necessary computational resources.

References

[1] C. Adams, S. Tavares, The structured design of cryptographically good s-boxes, J. Cryptol. 3 (1) (1990) 27–41.
[2] M. Matsui, On Correlation Between the Order of S-boxes and the Strength of DES, Springer, Berlin, Heidelberg, 1995.
[3] E.F. Brickell, J.H. Moore, M.R. Purtill, Structure in the s-boxes of the des, Proceedings on Advances in Cryptology – CRYPTO '86 (1987) 3–8.
[4] J. Daemen, V. Rijmen, The Design of Rijndael: AES – The Advanced Encryption Standard, Springer, 2013.
[5] M. Matsui, A. Yamagishi, A new method for known plaintext attack of feal cipher, Workshop on the Theory and Application of Cryptographic Techniques (1992) 81–91.
[6] E. Biham, A. Shamir, Differential cryptanalysis of des-like cryptosystems, J. Cryptol. 4 (1) (1991) 3–72.
[7] E.Y. Xie, C. Li, S. Yu, J. L, On the cryptanalysis of Fridrich's chaotic image encryption scheme, Signal Process. 132 (2017) 150–154.
[8] C. Carlet, On the higher order nonlinearities of Boolean functions and s-boxes, and their generalizations, Proceedings SETA 2008 (2008) 345–367.
[9] C. Carlet, Vectorial Boolean functions for cryptography Boolean Models and Methods in Mathematics, Computer Science, and Engineering, 134, 2010, pp. 398–469.
[10] E.C. Laskari, C.M. Meletiou, M.N. Vrahatis, Utilizing evolutionary computation methods for the design of s-boxes, Proceedings International Conference on Computational Intelligence and Security 2006 (2006) 1299–1302.
[11] W. Millan, L. Burnett, G. Carter, A. Clark, E. Dawson, Evolutionary heuristics for finding cryptographically strong s-boxes, Lect. Notes Comput. Sci. 1726 (1999) 263–274.
[12] J.A. Clark, J.L. Jacob, S. Stepney, The design of s-boxes by simulated annealing, New Gener. Comput. 23 (3) (2005) 219–231.
[13] B.A. Huberman, R.M. Lukose, T. Hogg, An economics approach to hard computational problems, Science 27 (1997) 51–53.
[14] C.P. Gomes, B. Selman, Algorithm portfolio design: theory vs. practice, Proceedings Thirteenth Conference on Uncertainty in Artificial Intelligence (1997) 190–197.
[15] M.-L. Cauwet, J. Liu, B. Rozire, O. Teytaud, Algorithm portfolios for noisy optimization, Ann. Math. Artif. Intell. (2015) 1–30.
[16] S.Y. Yuen, X. Zhang, On composing an algorithm portfolio, Memet. Comput. 7 (2015) 203–214.
[17] S.Y. Yuen, C.K. Chow, X. Zhang, Y. Lou, Which algorithm should I choose: an evolutionary algorithm portfolio approach, Appl. Soft Comput. 40 (1) (2016) 654–673.
[18] A. Almakhlafi, J. Knowles, Systematic construction of algorithm portfolios for a maintenance scheduling problem, Proceedings CEC 2013 (2013) 245–252.
[19] D. Souravlias, K.E. Parsopoulos, E. Alba, Parallel algorithm portfolio with market trading-based time allocation, Proceedings International Conference on Operations Research 2014 (OR2014) (2014) 567–574.
[20] S.Y. Yuen, X. Zhang, Multiobjective evolutionary algorithm portfolio: choosing suitable algorithm for multiobjective optimization problem, Proceedings CEC 2014 (2014) 1967–1973.
[21] F. Peng, K. Tang, G. Chen, X. Yao, Population-based algorithm portfolios for numerical optimization, IEEE Trans. Evol. Comp. 14 (5) (2010) 782–800.
[22] J.A. Vrugt, B.A. Robinson, J.M. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Trans. on Evol. Comput. 13 (2) (2009) 243–259.

[23] W. Millan, Smart hill climbing finds better Boolean functions, Proceedings Boolean Functions Workshop on Selected Areas on Cryptography, SAC 97 (1997) 50–63.
[24] F. Glover, Future paths for integer programming and links to artificial intelligence, Comput. Oper. Res. 13 (5) (1986) 533–549.
[25] S. Kirkpatrick, C.D. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science 220 (4598) (1983) 671–680.
[26] M. Gendreau, J.-Y. Potvin, Tabu search, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, Springer, 2010, pp. 41–60.
[27] A.G. Nikolaev, S.H. Jacobson, Simulated annealing, in: M. Gendreau, J.-Y. Potvin (Eds.), Handbook of Metaheuristics, Springer, 2010, pp. 1–40.
[28] N. Shukla, Y. Dashora, M. Tiwari, F. Chan, T. Wong, Introducing algorithm portfolios to a class of vehicle routing and scheduling problem, Proceedings OSCM 2007 (2007) 1015–1026.
[29] N. Shukla, A. Choudhary, P. Prakash, K. Fernandes, M. Tiwari, Algorithm portfolios for logistics optimization considering stochastic demands and mobility allowance, Int. J. Prod. Econ. 141 (2013) 146–166.
[30] D. Souravlias, K.E. Parsopoulos, I.S. Kotsireas, Circulant weighing matrices: a demanding challenge for parallel optimization metaheuristics, Optim. Lett. 10 (6) (2016) 1303–1314.
[31] R.C. Fukunaga, J. Kennedy, Genetic algorithm portfolios, Proceedings CEC 2000 2 (2000) 1304–1311.
[32] S. Picek, J.F. Miller, D. Jakobovic, L. Batina, Cartesian genetic programming approach for generating substitution boxes of different sizes, Proceedings GECCO 2015 (Companion Volume) (2015) 1457–1458.
[33] K. Nyberg, Perfect nonlinear s-boxes, Proceedings Workshop on the Theory and Application of Cryptographic Techniques (1991) 378–386.
[34] C. Carlet, D. Cunsheng, Nonlinearities of s-boxes, Finite Fields Appl. 13 (1) (2007) 121–135.
[35] J.A. Clark, J.L. Jacob, S. Stepney, Searching for cost functions, Proceedings CEC 2004 (2004) 1517–1524.
[36] E. Alba, G. Luque, S. Nesmachnow, Parallel metaheuristics: recent advances and new trends, Int. Trans. Oper. Res. 20 (1) (2013) 1–48.
[37] S. Picek, M. Cupic, L. Rotim, A new cost function for evolution of s-boxes, Evol. Comput. 24 (4) (2016) 695–718.
[38] W. Millan, How to improve the non-linearity of bijective s-boxes, Proceedings 3rd Australian Conference on Information Security and Privacy (1998) 181–192.
[39] J. Fuller, W. Millan, Linear redundancy in s-boxes, Proceedings FSE 2003 (2003) 74–86.
[40] G. Ivanov, N. Nikolov, S. Nikova, Reversed genetic algorithms for generation of bijective s-boxes with good cryptographic properties, Cryptogr. Commun. 8 (2) (2016) 247–276.
[41] Y. Nawaz, K.C. Gupta, G. Gong, Algebraic immunity of s-boxes based on power mappings: analysis and construction, IEEE Trans. Inf. Theory 55 (9) (2009) 4263–4273.