Computers & Industrial Engineering 136 (2019) 591–613

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

A novel optimization booster algorithm

S.H. Pakzad-Moghaddam*, Hassan Mina, Parisa Mostafazadeh

School of Industrial Engineering, College of Engineering, University of Tehran, Tehran, Iran

ARTICLE INFO

Keywords:
Artificial intelligence
Computer science
Evolutionary computation
Mathematical optimization
Exchange market

ABSTRACT

In this paper, a novel meta-heuristic method called the Optimization Booster Algorithm (OBA) is presented. It incorporates existing optimization methods with human-inspired intelligence, which applies particularly while conducting business in exchange markets. A key objective in exchange markets is to increase wealth over time, which is typically the same objective when performing optimizations. Moreover, optimization is about finding a way to increase the fitness value of a system, by spending adequate computation time. The OBA is founded on the core idea that a key reason behind the rapid evolution of human societies compared to the tortoise-like natural evolution is the forward-looking approach. In exchange markets, analysts have learned to make decisions based on forecasted prices, rather than the current prices; and this is an illustrative example of the application of such forward-looking approaches, which form the essence of the OBA. Following extensive numerical experiments, and applications of fourteen well-known heuristic and meta-heuristic methods to solve seventy-one non-linear unconstrained and constrained, single-objective and multi-objective benchmarks, before and after receiving a boost, the OBA performance is investigated. It has proven — in both theory and practice — to quite significantly improve existing optimization methods. In most cases, boosting resulted in much better quality of outputs, while requiring less computation time.

1. Introduction

Engineering optimization deals with finding optimal solutions for a given engineering problem (Mirjalili, Jangir, Mirjalili, Saremi, & Trivedi, 2017). Optimization methods are segmented as either exact methods or heuristic methods. In practice, though, suitable types of optimization methods are chosen with regards to a problem's characteristics, such as the problem size (Puchinger & Raidl, 2005). The most popular exact optimization methods are branch-and-bound (Lawler & Wood, 1966), Benders' decomposition (Fisher, 1985), dynamic programming (Karlin, 1955), and linear and integer programming-based methods, such as branch and price, and branch and cut methods (Wolsey & Nemhauser, 2014).

Heuristic methods can be categorized into two groups (Focacci, Laburthe, & Lodi, 2003): construction algorithms (Brusco, 2015; Laha & Gupta, 2016; Pakzad-Moghadam, Shahmohammadi, & Ghodsi, 2012) and local search methods. Local search methods repeatedly try to improve on current solutions; and when a better solution is found, it replaces the former solutions and continues with subsequent iterations until no better solution can be found. However, a major drawback of these methods is the risk of becoming stuck in local optima, which can be dealt with by applying meta-heuristics. These algorithms are often associated with concepts from different fields of research, such as evolution, swarm dynamics, physics and human behavior (Mirjalili, 2016).

Most meta-heuristic algorithms are inspired by natural behavioral phenomena (Salimi, 2015). Several meta-heuristic algorithms, such as the Evolution Strategy (ES) (Rechenberg, 1978), the Memetic Algorithm (MA) (Moscato, 1989), the Genetic Algorithm (GA) (Holland, 1992), the Differential Evolution (DE) algorithm (Storn & Price, 1997), the Symbiotic Organisms Search (SOS) (Cheng & Prayogo, 2014), the Farmland Fertility (Shayanfar & Gharehchopogh, 2018), are inspired by rules of natural evolution. These are collectively termed evolution-based algorithms. These meta-heuristics have been extensively applied to solve computationally hard optimization problems (Gao, Zhang, Zhang, & Li, 2011; Guo & Tang, 2019; Liao, Fan, & Li, 2019).

Swarm intelligence optimization algorithms generally use reduced mathematical models of the complex social behavior of insects and animal groups (Shareef, Ibrahim, & Mutlag, 2015); among which, the Particle Swarm Optimization (PSO) (Eberhart & Kennedy, 1995) is the most popular. Other swarm-based algorithms are the Ant Colony Optimization (ACO) (Dorigo, 1992; Dorigo, Maniezzo, & Colorni, 1991), the Cuckoo Search (CS) (Yang & Deb, 2009), the Fruit Fly Optimization Algorithm (FOA) (Pan, 2012), the Animal Migration Optimization

* Corresponding author.
E-mail address: hpakzad@ut.ac.ir (S.H. Pakzad-Moghaddam).

https://doi.org/10.1016/j.cie.2019.07.046
Received 22 February 2019; Received in revised form 1 June 2019; Accepted 26 July 2019
Available online 01 August 2019
0360-8352/

(AMO) (Li, Zhang, & Yin, 2014), the Moth-Flame Optimization (MFO) (Mirjalili, 2015), the Lion Optimization Algorithm (LOA) (Yazdani & Jolai, 2016), the Whale Optimization Algorithm (WOA) (Mirjalili & Lewis, 2016), the Multi-Objective Bee Foraging Algorithm (MOBFA) (Ma et al., 2017), the Rhino Herd (RH) (Wang, Gao, Zenger, & Coelho, 2018), and the Pity Beetle Algorithm (PBA) (Kallioras, Lagaros, & Avtzis, 2018). This type of meta-heuristics also has played a significant role in solving optimization problems around the globe (Jia et al., 2018; Kaasgari, Imani, & Mahmoodjanloo, 2017; Laha & Gupta, 2018; Pakzad-Moghaddam, 2016; Shishegar, Duchesne, & Pelletier, 2018).

Physics-based methods are inspired by the rules of physics, which include Simulated Annealing (SA) (Kirkpatrick, Gelatt, & Vecchi, 1983), the Central Force Optimization (CFO) (Formato, 2007), the Spiral Dynamics Inspired Optimization (SDIO) (Tamura & Yasuda, 2011), the Optics Inspired Optimization (OIO) (Kashan, 2015), and the Find-Fix-Finish-Exploit-Analyze (F3EA) (Kashan, Tavakkoli-Moghaddam, & Gen, 2019). Following are some examples for the application of physics-based meta-heuristics to optimize search problems (Ceschia, Di Gaspero, & Schaerf, 2017; Madani-Isfahani, Tavakkoli-Moghaddam, & Naderi, 2014; Marandi & Ghomi, 2019).

The final category of meta-heuristic methods involves those that are inspired by human intelligence, where the most well-known and practical of the human-based algorithms are the Tabu Search (TS) (Glover, 1989; 1990), the Harmony Search (HS) (Geem, Kim, & Loganathan, 2001), the Imperialist Competitive Algorithm (ICA) (Atashpaz-Gargari & Lucas, 2007), the Teaching Learning Based Optimization (TLBO) (Rao, Savsani, & Vakharia, 2011) and the Social Engineering Optimizer (Fathollahi-Fard, Hajiaghaei-Keshteli, & Tavakkoli-Moghaddam, 2018). Plenty of researchers such as (Kiziloz & Dokeroglu, 2018; Pakzad-Moghaddam, Mina, & Tavakkoli-Moghaddam, 2014; Pakzad-Moghaddam, Mina, Iranmanesh, & Keyvandarian, 2015) has wielded human intelligence-inspired algorithms to approach combinatorial and NP-complete optimization problems in particular.

The literature review shows that there is a scarcity in human-inspired artificial intelligence, though the term artificial should not be misleading as only considering non-human inspirations. In fact, comparing swiftly-evolving human societies with the tortoise-like evolution of natural ecosystems establishes human intelligence as a very rich source of inspiration. One major advantage of human intelligence that can be applied, when facing highly complex socio-economic problems, is its forward-looking approach. The OBA has been designed to equip optimization algorithms with such forward-looking intelligence, in order to improve their overall performances.

The proposed OBA is applied over several steps that are described in detail in the next section; and after a detailed overview, the capability of the OBA in improving the performance of existing optimization methods is investigated and proven mathematically. Various benchmark optimization functions, as well as real-world design engineering problems and Travelling Salesman Problems (TSPs), are then used as benchmarks in Sections 3 and 4, where the efficiency gains of the OBA are shown numerically by comparing the performances of fourteen well-known optimization methods, before and after receiving a boost. Finally, Section 5 presents the concluding remarks, and possible avenues for future studies.

2. The optimization booster algorithm

2.1. Overview

Over the years, many theoretical and experimental methods have been perfected to allow intelligent decisions to be made while operating in markets. These methods are, in essence, optimization techniques that have been applied to find the best choices of investments to maximize profits. The question is, though, how to take advantage of such intelligence, which has been enriched by the human mind over years of experience, so as to improve the performance of current optimization methods. To this end, the Optimization Booster Algorithm has been developed. Yet, before discussing the OBA, it is important to consider the fundamental similarities that exist between exchange markets and optimization problems.

An exchange market provides numerous choices of investment (such as portfolios) that include every possible combination of different securities, and at all different budgets. In the optimization literature, the solution space contains solutions that are combinations of the different values of decision variables, which have been subjected to particular sets of constraints. An exchange market is therefore a market that is organized for the purpose of making a profit; and traders attempt to increase their wealth by buying and selling securities over the course of time. Scientists and engineers on the other hand, use optimization methods to attempt to improve their objective values, by trading non-optimal decision values with optimum ones. In other words, traders aim to increase their capital over time, and optimization search methods attempt to improve the fitness values of their objective functions at a cost of computation time.

As the market is partitioned into different segments (such as the food, electronics, steel and medicine sectors), for the purpose of simplifying and increasing the agility of the OBA's decision making process, the same is done by creating neighborhoods throughout the solution space. A bull, or booming market sector is a sector in which prices are either rising, or expected to rise; thus, with the OBA, neighborhoods with a higher probability of finding better solutions are termed bull neighborhoods. Market sectors in which prices are falling are known as bear, or down sectors; so, with the OBA, neighborhoods with lower probabilities of finding better solutions are termed bear neighborhoods.

The key to becoming successful in exchange markets is knowing the right time to buy or sell securities, where the boom-and-bust cycle is a key characteristic in free economies. This is illustrated by a frequent process of economic expansion and contraction. Obviously, to maximize profits, booms and busts should be anticipated to allow profitable securities to be bought before booms, and sold before busts. Since future trends are never certain, an important question, then, is how to anticipate booms and busts as accurately as possible. Gaining experience and being well-trained remarkably enhances the chances of acting more successfully in this process, which is why experienced traders often earn more profits than novice traders.

Now, let us imagine that a trade optimizer is surfing an objective surface through the boundaries of a solution space. Gaining inspiration from professional traders, the optimizer should be able to better forecast booms and busts, enabling the optimizer to find the best possible solutions with the least time and effort. This is where the OBA is able to help an optimizer anticipate what will happen, without having to spend a great deal of time surfing through the entire solution space.

During optimization, there is a high risk of becoming trapped in local optima while performing trials to figure out which way to proceed. With the OBA, the idea is to measure the likelihood of further improvements before starting to search different neighborhoods. In this way, an optimizer has the ability to speculate what will happen if it continues to search a neighborhood; whereby, applying the OBA helps the optimizer to distinguish bull neighborhoods from bear neighborhoods with considerably lower computational efforts. Continuing to search in the most likely of bull neighborhoods is exactly the same task that a professional trader performs in exchange markets. To this end, the following steps are performed iteratively in the OBA:

1- Macro searching between neighborhoods to find bull neighborhoods; and
2- Micro searching within selected bull neighborhoods to find optimum solutions.

Hence, instead of searching for the best solution from the whole solution space, the OBA seeks to find the best neighborhoods containing the highest chances of optimal solutions (bull neighborhoods). Then,

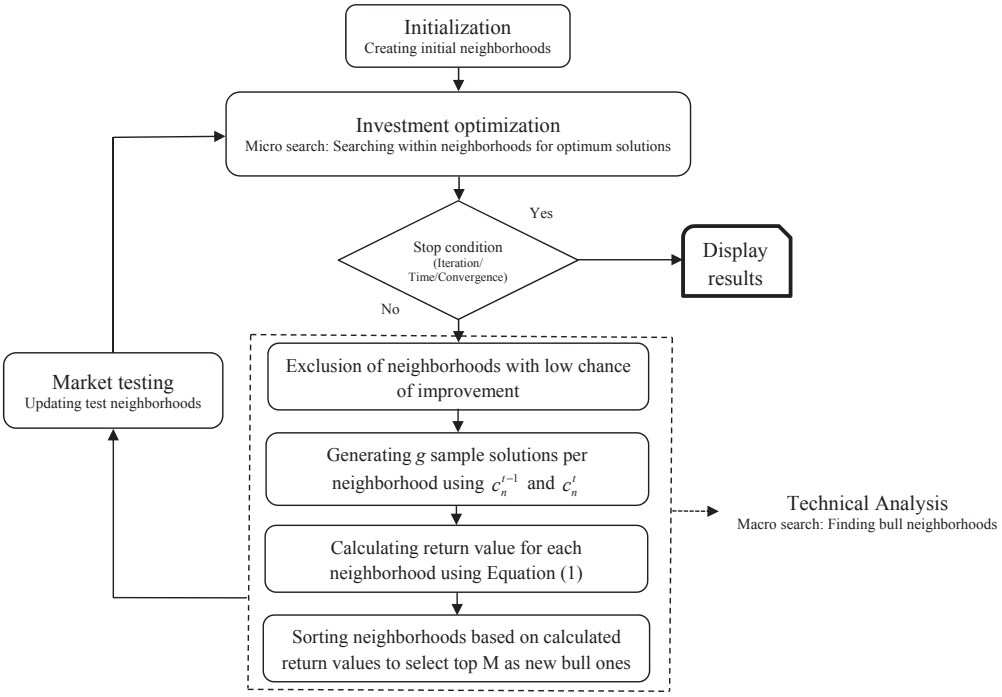


Fig. 1. Flowchart of OBA.

the solution space is temporarily reduced to the selected neighborhoods, and a boost is applied to the optimization methods to search selected solution sub-spaces.

The boost mechanism of the OBA is investigated analytically in Section 2.3. Considering this two-phase process of the algorithm, it has been mathematically proven here that boosting an optimization method most likely improves its performance; or at the least leaves it unchanged. Identifying those neighborhoods with the highest and lowest improvement chances, before activating an optimization method, is therefore as important as recognizing bull and bear sectors before making an investment.

2.2. Overview of the algorithm

A flowchart of the OBA is presented in Fig. 1.

a. Initialization

The first step in population-based optimization algorithms is always initializing a population. The OBA smart search is no exception; and it, too, starts with an initialization step. The difference here, though, is that the population in the OBA is not a set of single solutions, but rather the set of neighborhoods each covering either an infinite or limited set of solutions. The outcome of this step is that K random solutions are generated as the center of K randomly chosen neighborhoods.

b. Investment optimization

Next, it is time to search for optimality within each and every neighborhood. The optimization tool that is used to search neighborhoods is, in fact, the optimization method that receives a boost. Thus, two search mechanisms are executed iteratively while searching the solution space with a boosted algorithm. The first is the macro search between neighborhoods to find bull neighborhoods, and the OBA takes care of this part through the next step (the technical analysis). The second mechanism which is the investment optimization is the micro search within the selected neighborhoods, to spot the optimum solutions via the optimization algorithm that is being boosted. Consequently, the outcome of the investment optimization step is K local optimum points forming the center of a new set of updated neighborhoods (c_n^0, $n = 1, 2, \ldots, K$).

c. Technical analysis

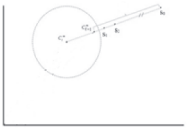
Through this step, an analytical approach is employed in order to select M bull neighborhoods, where $M \leq K$. Therefore, the outcome of this step is a chosen set of M bull neighborhoods with the highest chances of further improvement. This implies that the OBA takes advantage of a forward-looking approach, instead of concentrating on present data. The chosen bull neighborhoods should be those with the highest improvement likelihoods, and not necessarily those covering solutions with the highest objective values. The remaining neighborhoods, which are not likely to contain undiscovered solutions with better objective values, are called bear neighborhoods. Thus, inspired by the common behaviors of professional traders, which are to avoid investing in markets that historically have been proven to be declining, neighborhoods with no historical improvements are considered to be bear neighborhoods. The historical improvement of a neighborhood is measured by comparing the fitness of the current center of the neighborhood ($f(c_n^t)$) with that of the previous neighborhood ($f(c_n^{t-1})$), where c_n^t is the current center of the n^{th} neighborhood; $c_n^{t-1}$is the previous center; and f is the fitness function.

After excluding bear neighborhoods, the fitness of the remaining neighborhoods is measured in terms of the expected future improvements, not the current level of optimality within the neighborhoods. This is inspired by looking into the attractiveness measures of the exchange markets, which are based on expected future return on investments, rather than current price levels. The function that is applied for this purpose is therefore a return function that is inspired by smart investment strategies, where the return function is aimed at providing a projection on the likelihood of a neighborhood containing better solutions.

In order to calculate the return value of a neighborhood, g forward-looking sample solutions are generated within that neighborhood, in a way that extending the vector that connects the previous center of the neighborhood (c_n^{t-1}) to the new one (c_n^t) will also result in crossing these

sample solutions as well. Eventually, the calculated return value for the new neighborhood is a linear combination of the center's objective value and the maximum objective value that is calculated for these forward-looking sample solutions.

Let us assume that s_n^0, s_n^1, …, s_n^g are the generated sample solutions for a neighborhood, n. The maximums of $f(s_n^0), f(s_n^1), \ldots, f(s_n^g)$ are used as the projections on the future optimality within this neighborhood; hence, the maximum $\{f(s_n^i)|$ where $i = 1, 2, \ldots, g\} - f(c_n^t)$ will be the maximum expected growth as a result of investing enough computation time to explore the n^{th} neighborhood. The final return value is then a weighted summation of the calculated expected growth, and the center's objective value, which is defined in order to take the expected growth, as well as the currently discovered optimality of the neighborhood into account. Considering η as the weight applied for calculating the return value, which is called the vista coefficient, the applied return function would be as follows:



$$\tau_n^t = \eta \times (max\ \{f(s_n^i) - f(c_n^t)\}) + f(c_n^t) = \eta \times max\ \{f(s_n^i)\} + (1-\eta) \times f(c_n^t) \tag{1}$$

where τ_n^t is the return value calculated for the neighborhood, n, in iteration t. The next step is to prioritize the neighborhoods, and pick M bull neighborhoods with the highest return values.

d. Market testing

In order to augment the global search abilities of the OBA, K new test neighborhoods are generated during this step. A formula called the trade function is applied in order to update the test neighborhoods that have been generated in the previous iteration, which is the same as trading the previous test portfolios with new ones. The updated trade mechanism is based on taking updated knowledge about the solution space or exchange market, as well as contained bull neighborhoods or sectors, into consideration.

$$p_k^t = p_k^{t-1} + \lambda \times \vec{\varepsilon} \times \frac{\sum_{m=1}^{M}(c_m^t - p_k^{t-1})}{M}$$
$$\varepsilon_j \approx U[0\ \ 1] \qquad j = 1,\ 2,\ \ldots,\ J \tag{2}$$

where t represents the iteration number; J is the dimension of the solution space; $\vec{\varepsilon}$ is a vector of J uniformly distributed random probabilistic variables; p_k^t is the k^{th} test solution selected in the iteration, t, as the center of k^{th} test neighborhood; and λ is the spread coefficient that is applied to control the diversity of the test neighborhoods. As an example, the tail of 20 test solutions in 40 iterations, where M equals 5 and λ is increasing from 0.1 to 4, is depicted in Fig. 2. Clearly, increasing λ results in wider covered spaces.

The outcome of market testing is that K new updated neighborhoods will be added to M current bull neighborhoods. All these M + K neighborhoods will be searched more intensively through the investment optimization step; among which, M new bull neighborhoods will then be discovered while performing the technical analysis. This closed loop then continues until the stop condition is met; whereby, the OBA stops searching when either the number of iterations exceeds a predefined limit; or there is no more time left for execution; or it is observed to have been converged using a predetermined convergence threshold. This last stop condition is the stop condition that has been applied in Sections 3 and 4.

2.3. Boost impact

An optimization algorithm, X, is selected here to look for the optimum value throughout the search domain, R, to consider a very simplified form of the OBA, as follows:

1. Macro search: Selecting a neighborhood;
2. Micro search: Using X to search the selected neighborhood.

To find the global optimum solution within the solution space, the two following cases shall occur for the Boosted X(BX).

1. A: The selected neighborhood contains the global optimum.
2. B: The global optimum is found while searching the selected neighborhood via X.

Assuming $r \leq$ R to be the search domain of the selected neighborhood, the probability of A occurring is μ_r; while p_r is the probability of B occurring. Therefore, the probability of finding the global optimum via BX equals the product of μ_r and p_r, as follows:

$$P_r = \mu_r \times p_r \ \Rightarrow \quad Q_r = 1 - \mu_r \times p_r \tag{3}$$

where P_r and Q_r represent the probabilities of BX finding or not finding the global optimum, respectively, while exploring a single neighborhood. In the case where n neighborhoods are being selected independently (randomly, as opposed to intelligently), Q_r would be calculated as follows:

$$Q_r = (1 - \mu_r \times p_r)^n \tag{4}$$

Now, let us assume that r^* is the optimal search domain, in order to minimize Q_r.

$$Q_{r^*} \leqslant Q_r \Rightarrow Q_{r^*} \leqslant (1 - \mu_r \times p_r)^n \quad \forall\ 0 \leqslant r \leqslant R \tag{5}$$

$$r = R \Rightarrow Q_{r^*} \leqslant (1 - \mu_R \times p_R)^n \tag{6}$$

$$\mu_R = 1 \Rightarrow Q_{r^*} \leqslant (1 - p_R)^n \Rightarrow 1 - P_{r^*} \leqslant (1 - p_R)^n \leqslant 1 - p_R \Rightarrow P_{r^*} > > p_R \quad \forall\ n > 0 \tag{7}$$

Therefore, it has been mathematically proven, here, that the optimization algorithm, X, is outperformed by its boosted version, BX, when associated with the optimal neighborhood search domain, r^*, and at least one selected neighborhood ($n > 0$).

Furthermore, the more intelligent the neighborhood selection is, so too is μ_r increased. Instead of picking random neighborhoods independently, therefore, human intelligence inspirations have been employed in the OBA to enhance μ_r. Applying return values instead of absolute fitness values, looking into historical improvements within a neighborhood, and updating the test neighborhoods using a trade function, results in maximizing μr. In fact, the OBA iteratively uses intelligence that is obtained from previous iterations to increase μ_r. Thus, as has been mathematically proven, boosting optimization methods is expected to improve performance, even with a random neighborhood selection. In the next sections, experimental results also illustrate the OBA's excellent performance in improving the quality of solutions, while consuming considerably less computation time.

3. Experimental studies

Next, a variety of optimization methods were chosen to cover a wide spectrum of techniques, including heuristics and meta-heuristics; differentiable and non-differentiable; single-objective and multi-objective; population-based and trajectory-based; and global and local search techniques. The performance of the OBA is reported, here, while

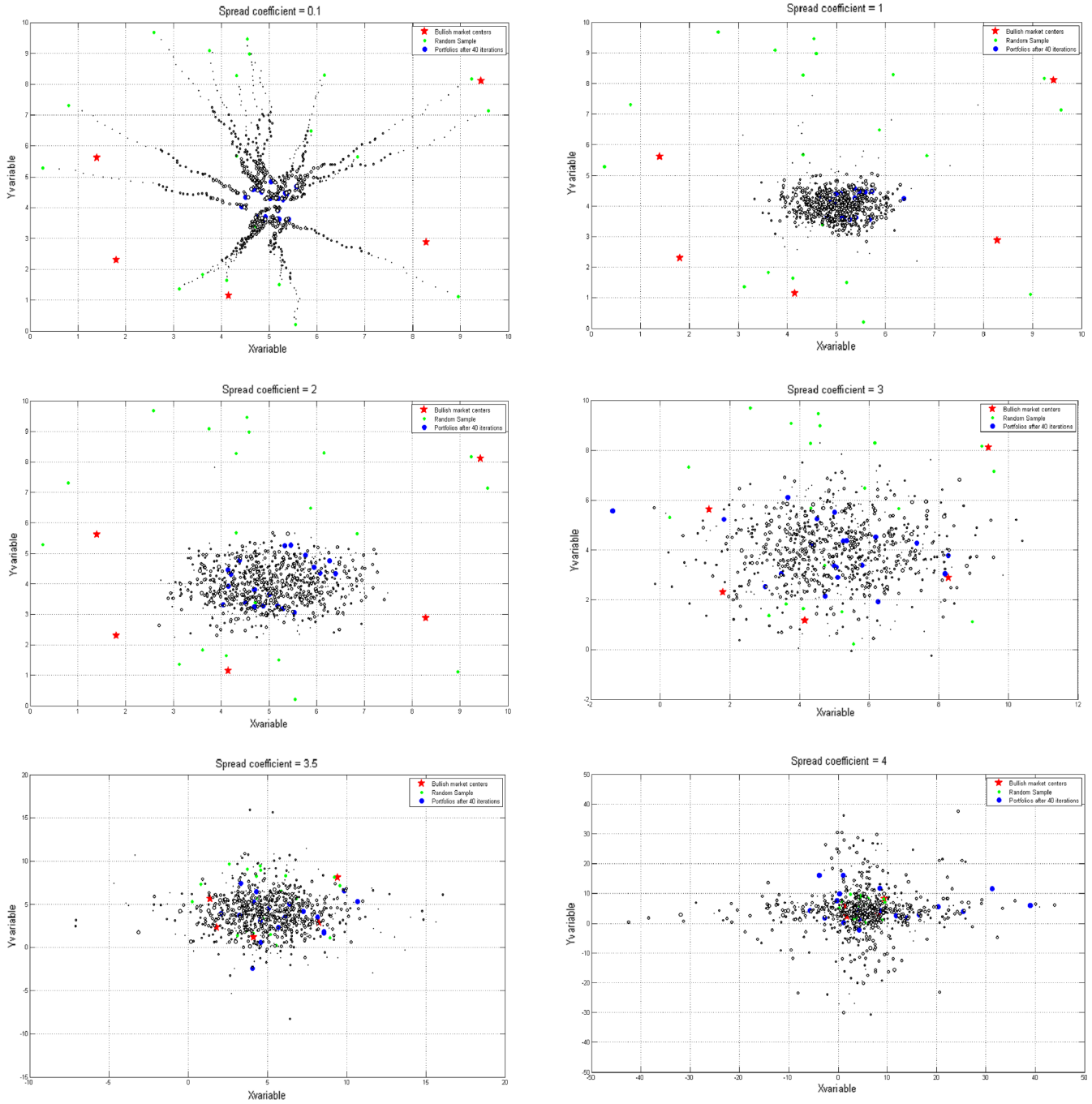


Fig. 2. Market testing results in 40 iterations for $0.1 \leqslant \lambda \leqslant 4$.

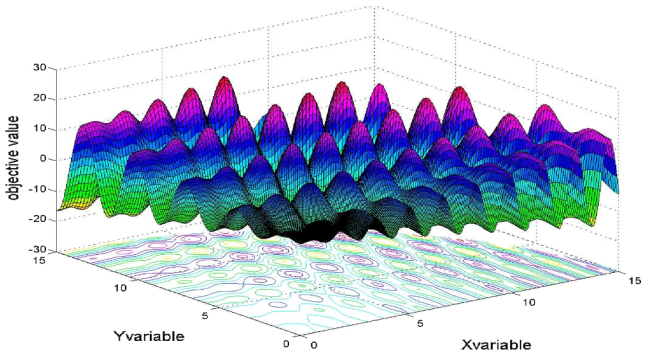


Fig. 3. The Benchmark function in 3-dimension view.

boosting these optimization techniques for solving continuous/combinatorial constrained/unconstrained non-linear benchmarks.

3.1. Validation

First, a benchmark function was applied with numerous local optima (Yao, Liu, & Lin, 1999). The function has been presented in the polar coordinate system, and plotted in three-dimensional form in Fig. 3.

$$f(r, \theta) = r\cos(\theta) \times \sin(4r\cos(\theta)) + 1.1 \times r\sin(\theta) \times \sin(2r\sin(\theta)) \quad (8)$$

Where the exchange marker or solution space is as follows.

$$B = \left\{(r, \theta) | 0 \leqslant r \leqslant 15 \& 0 \leqslant \theta \leqslant \frac{\pi}{2}\right\} \quad (9)$$

Table 1
BSA parameters.

Algorithm	Parameters	Value
OBA	Number of test sectors (K)	20
	Grid points (g)	4
	Maximum number of bull sectors (M)	3
	Vista coefficient (η)	0.7
	Spread coefficient (λ)	3.2
SA	Initial temperature (T_0)	4
	Temperature reduction rate (α)	0.95
	Stop condition	Runtime = 0.1

Simulated Annealing (SA) was applied here as the optimization method that was to receive a boost. The parameters of the Boosted SA (BSA) are presented in Table 1. The centers of the bull neighborhoods in the polar coordinate system, obtained through BSA, are presented in Table 2.

BSA continued its intelligent search until the stop condition was met. Finally, the solutions of (14.8658, 0.9173) in the polar coordinate system, or (11.8029, 9.0379) in the Cartesian coordinate system were obtained as the centers of the first bull neighborhood in the second iteration. The mentioned solution was the global optimum for this benchmark function, with an objective value of −22.0062. BSA results are schematically outlined in Figs. 4–7.

Note that the polar coordinate system was applied here with the sole purpose of improving the quality of the graphical presentation. However, there is no obligation or preference to perform the applied coordinate system when boosting an algorithm.

3.2. Unconstrained non-linear benchmarks

Numerous high-dimensional benchmarks were applied in order to check the ability of the OBA to improve the performance of several heuristic and meta-heuristic methods. Some of these methods are designed to deal with large-scale optimization problems, such as the Trust Region Gradient-Based (TRGB), PSO and DE methods. In addition, multiple low-dimensional benchmarks were also applied in order to be solved by medium-scale optimization methods, such as the Broyde-Fletcher-Goldfrab-Shanno (BFGS) Quasi-Newton method. Many of these benchmark functions were adopted from "The Special Session on Real-parameter Optimization of the 2005 IEEE Congress on Evolutionary Computation" (CEC2005) (Suganthan, Hansen, Liang, Deb, Chen, Auger, & Tiwari, 2005) and "A literature survey of benchmark functions for global optimization problems" (Jamil & Yang, 2013). The rest were adopted from previous works by Yao et al. (1999), Karaboga and Akay (2009), He, Wu, and Saunders (2009), Rao and Patel (2013), and Kashan (2015).

All of the applied benchmark functions are presented in Table A.1; whereby, 53 low-to-high dimensional unconstrained benchmark functions have been presented. Gradient functions for the optimization problems that were to be optimized via the gradient-based methods are also provided in this table; while the two last columns of the table are devoted to representing the dimensions and global optima of these benchmarks. In most cases, the global optimum values of the functions

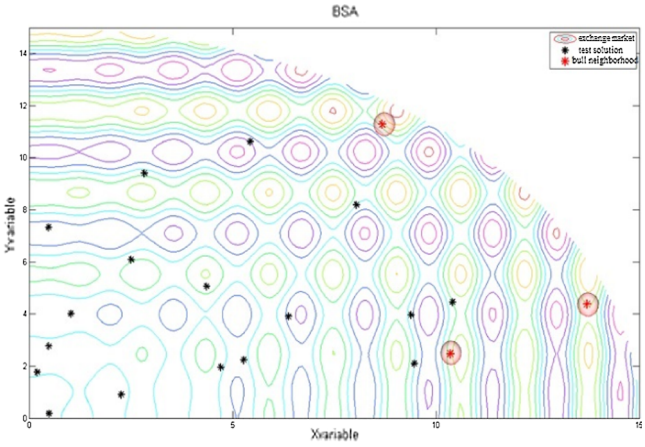


Fig. 4. Initialization.

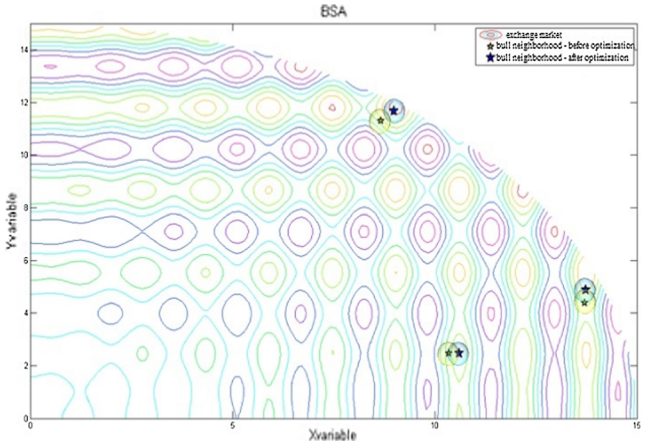


Fig. 5. Iteration one (Investment optimization).

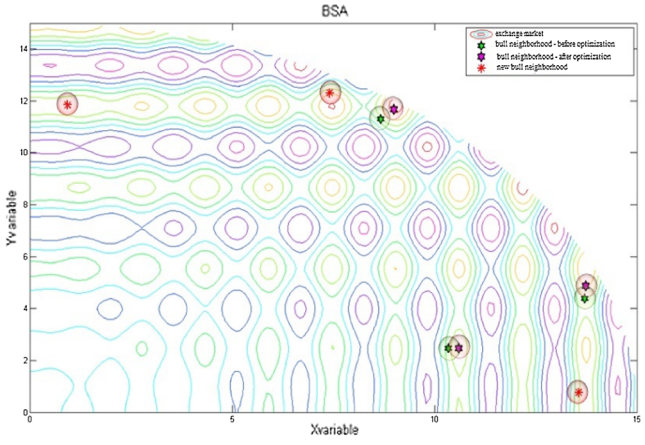


Fig. 6. Iteration two (Market testing).

Table 2
Center of bull neighborhoods.

Iteration	Bull market sectors					
	c_t^1	$f(c_t^1)$	c_t^2	$f(c_t^2)$	c_t^3	$f(c_t^3)$
Initialization	(14.24, 0.9153)	−8.13	(10.64, 0.2344)	−8.12	(14.40, 0.3086)	−10.61
1	(14.74, 0.9137)	−21.44	(10.89, 0.2275)	−13.25	(14.59, 0.3409)	−15.49
2	(14.8658, 0.9173)	−22.0062	(13.98, 1.0081)	−20.43	(14.76, 0.3732)	−19.51

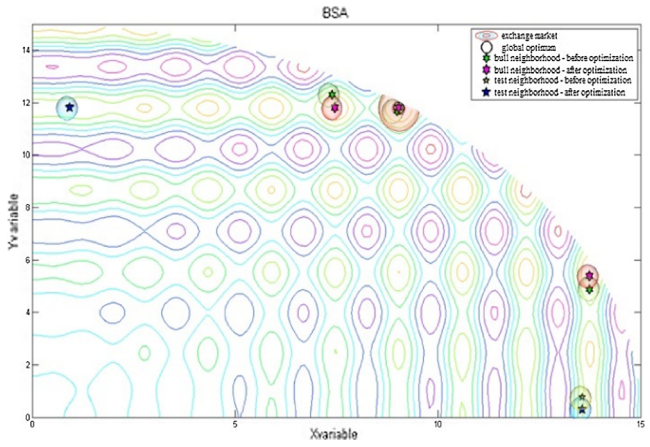


Fig. 7. Iteration two (Investment optimization).

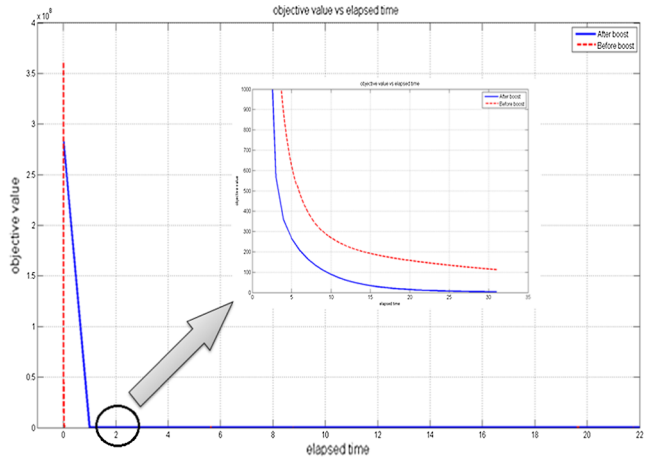


Fig. 8. Performance of steepest descent before and after receiving a boost.

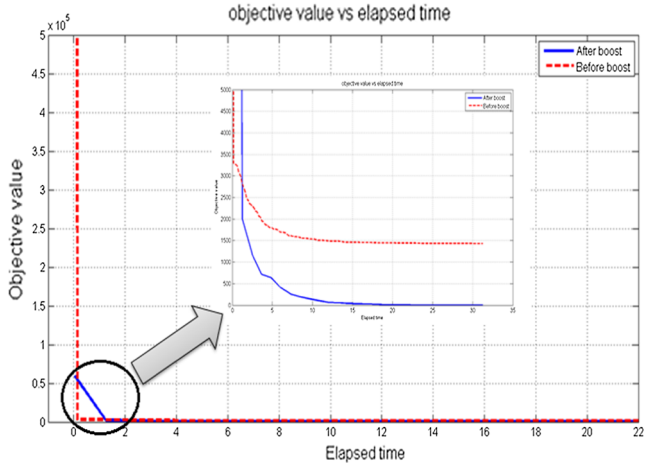


Fig. 9. Performance of Nelder-Mead before and after receiving a boost.

did not depend on their numbers of variables (dimensions); but in several cases, the global optimum values have been reported as the functions of the problems' dimensions (d).

- **Steepest descent and TRGB**

Both the methods of the steepest descent and the TRGB require analytically calculated gradients of a function to search for their optimum solutions. In this regard, gradient functions were provided for 22

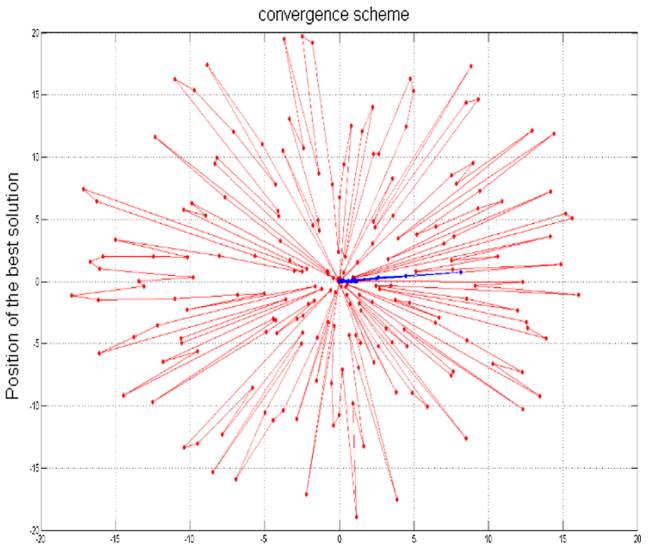


Fig. 10. Convergence scheme for boosted steepest descent solving F41.

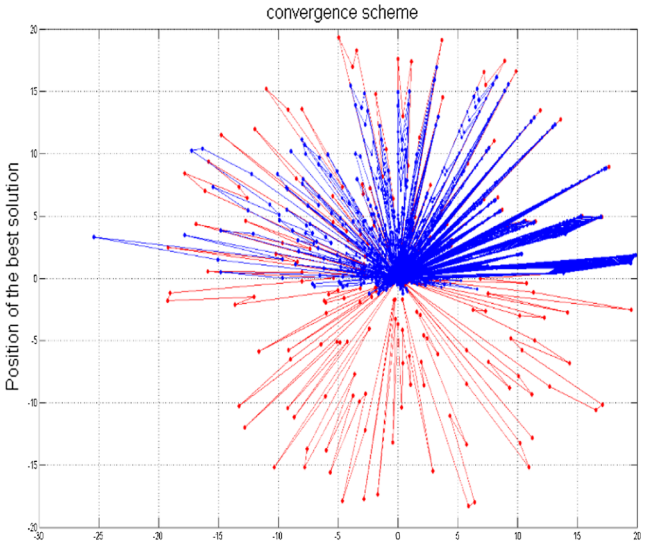


Fig. 11. Convergence scheme for steepest descent solving F41.

benchmarks, as depicted in the Table A.1. The steepest descent was coded by MATLAB 2012a, while the TRGB code was provided by its optimization toolbox, Optimtool. In order to eliminate the effects of the random initial solutions on the reported results, each of these benchmarks was solved 60 times using both methods. The obtained objective values were calculated for F41 versus the elapsed times consumed by both the steepest descent method, as well as its boosted version, and the values are plotted in Fig. 8.

The superiority of the boosted steepest descent can be easily observed on the magnified figure. It was observed that boosting the steepest descent enabled the algorithm to provide better solutions after only a few iterations. Figs. 10 and 11 show how fast it approached the global optimum of F41, $x_i = 0$.

Each point represents a decision variable, and its distance from the center point (0 0) demonstrates its value, where the solutions have been displayed by plotting the closed loops encompassing the center points. The first initial solution has been plotted in a red color, while the color blue has been applied to show the updated solution after one iteration. Due to the rapid solution improvement in the boosted version, the best solution that was found in the first iteration occupied a much smaller space around the center point compared to the initial solution. In fact,

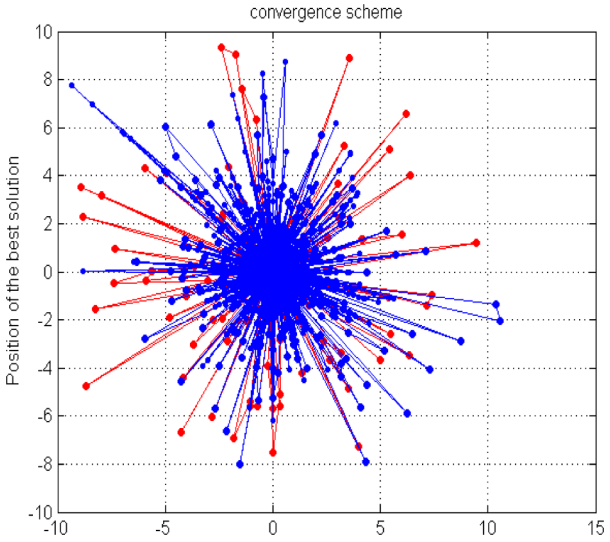


Fig. 12. Convergence scheme for boosted Nelder-Mead solving F48.

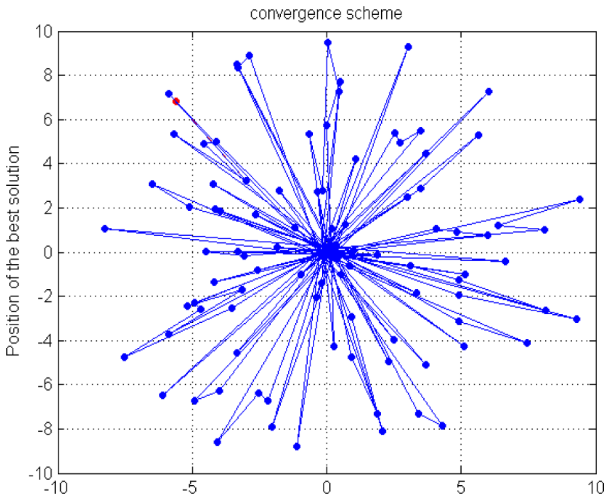


Fig. 13. Convergence scheme for Nelder-Mead solving F48.

comparing the red and blue colored diagrams in Figs. 12 and 13, it reveals that an impressive improvement was made in the performance of the steepest descent as a result of boosting (see Table 3 for more details).

The means and standard deviations of the optimum objective values and runtimes are reported in Table 3, which depicts the significant improvements that were observed in the obtained optimum values, as well as the considerable decreases that were also observed in the respective computation times, as a result of applying the OBA. When comparing the performances of each method to its boosted version, the better objective values and their runtimes are shown in bold; whereby, the total number of bold elements in each column has been provided in the last row as a quick comparison. Boosting the method of the steepest descent enabled it to find better solutions in all 22 cases, while its required runtime was decreased in 12 of the 22 cases. In addition, boosting the TRGB method made it capable of outperforming its traditional version in 21 cases, and it was faster in 14 out of the 22 cases.

Gradient-based methods are typically fast, but their major drawbacks are in becoming stuck in local optima. However, applying the forward-looking OBA approach while updating neighborhoods equipped the boosted algorithms with the ability to escape the local optima. In fact, this was the reason that their capability to find optimum solutions was improved so remarkably, though the improvement made in their required runtimes was less noticeable.

- **Nelder-Mead and BFGS**

The Nelder–Mead or downhill simplex method takes advantage of the concept of a simplex. It is applied for solving problems for which derivatives may not be known. On the other hand, the BFGS is a gradient-based method that uses Hessians to find optimum solutions; but it does not require gradient information to be supplied through analytically calculated gradient functions. This information is derived instead by partial derivatives using a numerical differentiation method with finite differences. Both methods were provided here using the MATLAB optimization toolbox, Optimtool; whereby, 23 benchmark functions (see the Table A.1) were solved by these two methods and their boosted versions. The optimum values versus the elapsed times for both the boosted and traditional Nelder-Mead methods in solving F48, are plotted in Fig. 9, while their convergence schemes are presented in Figs. 12 and 13.

Just as in the case of the steepest descent method, after a few iterations, the optimum value found by the boosted algorithm achieved a lower bound compared to that of the traditional Nelder-Mead method. Fig. 13 shows that the traditional Nelder-Mead method converged so slowly that almost no improvement was observed after the first iteration. On the other hand, Fig. 12 demonstrates that considerable improvement was made by the boosted Nelder-Mead method, even from the first iteration. Detailed results of these two methods and their boosted versions are reported in Table 4.

Table 4 portrays the outstanding ability of the OBA to guide the optimization algorithms towards better solutions. Boosting the BFGS and Neleder-Mead methods enabled them to find better solutions in 20 and 22 out of the 23 cases; while runtimes were decreased in 18 and 13 of the cases, respectively. In conclusion, boosting these two algorithms made them capable of finding much better solutions, or act faster, or both.

As observed, the OBA was developed to be flexible enough to equip all types of optimization methods with human-inspired intelligence. Due to this flexibility, the OBA is capable of augmenting the strengths, and overcoming the weaknesses of traditional optimization methods. For instance, regardless of whether or not a traditional algorithm is capable of performing local and global searches, its boosted version is fully capable of performing both local and global searches together, at the same time. The global search ability of the boosted algorithms is due to their processes of exploring multiple neighborhoods; while bounding the optimization methods to search within neighborhoods allows them to concurrently perform local searches.

The applied parameter values in this set of tests were observed to fluctuate throughout the following intervals: $K \in [1 \quad 12]$, $M \in [1 \quad 4]$, $\eta \in [0.5 \quad 1]$ and $\lambda \in [0.5 \quad 4]$. The two following gauges have been defined to summarize the results that were obtained.

$$g_X^f = \frac{f_{BX} - f^*}{f_{BX} + f_X - 2f^*} \tag{10}$$

$$g_x^T = \frac{T_{bx}}{T_{bx} + T_x} \tag{11}$$

where X is the optimization algorithm in its traditional form and BX is its boosted version; f^* is the global optimum for the objective function within the desired boundaries; f_X and f_{BX} represent the optimum objective values found by X and BX, respectively; T_X and T_{BX} indicate the time consumed by X and BX to find these optimum values; and g_X^f and g_X^T are measures of the effects of boosting on the algorithm, X, in terms of its optimality and runtime performance, respectively.

Note that, since using Number of Function Evaluations (NFE) results in extra efforts being neglected, such as iteratively calling trade and return functions while boosting an optimization method, execution times were compared before and after receiving a boost. These gauges

Table 3
Steepest descent and TRGB methods versus their boosted versions.

Function number		Global optimum	Search domain/ dimension	Steepest descent method				Boosted Steepest descent method		
				Objective value		Runtime		Objective value		Runtime
				Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean
1	F01	0.00	10/2	11.63	67.36E−2	**0.06**	0.22E−2	**0.46**	9.63E−2	1.05
2	F41	0.00	20/200	112.32	1.13E+01	31.03	553.08E−2	**3.65**	3.96E−2	**28.37**
3	F03	0.00	5/2	22.03	9.46E−2	**0.35E−2**	0.01E−2	**0.14E−2**	3.25E−05	20.12E−2
4	F04	−24777.00	20/2	8.24E−15	1.81E+02	66.75E−2	0.76E−2	**−24777.00**	0.00	**20.24E−2**
5	F02	0.00	50/2	7.89E−31	2.84E−32	0.24	10.33E−2	**0.00**	0.00	**0.54E−1**
6	F29	0.00	500/100	4.72E−23	8.94E−25	6.63	111.58E−2	**0.86E−26**	7.08E−28	**4.00**
7	F20	−1.00	100/2	0.00	7.18E−2	0.05E−1	0.09E−2	**−6.06E−156**	28.73E−2	**0.24E−2**
8	F53	0.00	10/2	1.60	7.73E−2	1.09	0.33E−2	**6.4E−4**	2.66E−05	**0.60**
9	F30	−1.00	1/10	−0.86	0.05	**6.19E−5**	93.58E−7	**−1.00**	0.00	0.10
10	F05	0.00	50/2	3.17E−1	2.33E−2	0.62	2.89E−2	**0.30E−68**	1.48E−71	**0.55**
11	F06	−103.16E−2	25/2	−021.22E−2	0.02E−2	18.12E−2	1.39E−2	**−103.16E−2**	0.00	**3.19E−2**
12	F23	0.00	10/4	14999.00	2.45E+03	**37.33E−2**	3.14E−2	**239.51E−2**	2.67E−2	50.14E−2
13	F08	−200.00	32/2	−199.21	11.77E−2	**2.25E−2**	0.43E−2	**−199.99**	2.77E−06	10.13E−2
14	F09	−21914.18E−2	32/2	−1099.36E−1	1.43E+01	**5.54E−5**	24.28E−8	**−1956.29E−1**	710.11E−2	4.22E−1
15	F14	0.00	10/2	824.14E−2	120.13E−2	10.72E−1	0.53E−1	**0.00**	0.00	**9.02E−1**
16	F15	0.00	5/2	1115.74E−2	18.73E−2	**6.19E−5**	69.52E−8	**65.43E−2**	14.97E−2	100.15E−2
17	F16	0.00	5/2	1886.52E−2	99.02E−2	**60.11E−6**	15.96E−7	**37.36E−2**	1.75E−2	150.14E−2
18	F17	0.00	5/2	2716.53E−2	382.01E−2	**5.28E−6**	17.60E−7	**4.29E−2**	0.03E−2	10.24E−2
19	F19	0.00	5/2	18.94E−2	0.02	2.97E−2	0.23E−2	**0.47E−2**	0.05E−2	**1.02E−2**
20	F48	0.00	10/80	41.48	919.85E−2	**30.04**	83.21E−1	**21.51E−2**	1.88E−2	41.00
21	F37	0.00	10/5	261.24	3.52E+01	0.32E−2	0.01E−1	**63.97**	1599.69E−-2	**0.16E−2**
22	F21	0.00	10/2	7.56E−71	5.39E−71	4.04	51.17E−2	**5.86E−131**	0.00	**3.89**
Overall				0/22		10/22		**22/22**		**12/22**

Function number	Boosted Steepest descent method	R/d	Trust region gradient-based (large-scale) method				Boosted Trust Region Gradient-Based method			
	Runtime		Objective value		Runtime		Objective value		Runtime	
	Stdv.		Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.
1	33.76E−2	10/2	4901.56E−2	232.92E−2	**5.82E−2**	6.30E−3	**1.06E−14**	9.21E−16	10.52E−2	5.30E−03
2	32.24E−1	20/500	1.06E38	2.24E+37	476.09E−2	3.34E−1	**6.22E−40**	2.78E−41	**100.45E−2**	9.37E−02
3	4.07E−2	5/2	4743.47E−2	73.25E−2	**4.16E−2**	4.50E−3	**3.53E−3**	2.60E−05	45.28E−2	1.20E−01
4	5.04E−2	20/2	**−24777.00**	0	1.76E−2	2.70E−3	**−24777.00**	0.00	**0.29E−2**	7.00E−04
5	0.79E−2	50/2	1.55E−19	3.67E−21	1.41E−2	2.10E−3	**3.15E−30**	3.70E−32	**0.44E−2**	1.00E−04
6	13.71E−2	500/500	4.84E−16	2.90E−17	2429.54E−2	1.79	**50.21E−25**	7.46E−26	**0.72E−2**	1.40E−03
7	0.02E−2	100/2	0	8.54E−2	**87.23E−6**	0.00	**−1**	0.00	0.28E−2	1.00E−04
8	10.08E−2	10/2	110.68E−12	5.62E−12	0.28E−1	1.00E−4	**158.35E−25**	2.61E−26	**0.56E−2**	1.40E−03
9	2.04E−2	1/100	−56.07E−9	0.07E−1	**6.67E−5**	0.00	**−1.00**	0.00	0.27E−2	3.00E−4
10	6.68E−2	50/2	29.86E−2	10.77E−2	2.22E−2	6.00E−3	**75.54E−15**	1.14E−15	**0.24E−2**	1.00E−3
11	0.12E−1	25/2	−21.55E−2	0.43E−1	**0.22E−2**	1.30E−3	**−103.16E−2**	0.00	1.26E−2	1.10E−3
12	0.09E−1	10/4	36.76E−1	2.96E−1	9.98E−2	0.00	**34.79E−8**	2.24E−07	**0.31E−2**	8.00E−4
13	0.97E−2	32/2	−199.99	0.03E−2	6.02E−2	1.90E−3	**−200.00**	0.00	**0.58E−2**	3.00E−4
14	7.59E−2	32/2	−1680.52E−1	246.22E−2	1.32E−2	1.80E−3	**−1928.14E−1**	314.36E−2	**0.04E−1**	2.20E−3
15	16.59E−2	10/2	122.07E−9	3.27E−08	0.21E−1	5.10E−3	**59.08E−16**	5.54E−17	**0.34E−2**	1.50E−3

(*continued on next page*)

Table 3 (*continued*)

Function number	Boosted Steepest descent method	R/d	Trust region gradient-based (large-scale) method				Boosted Trust Region Gradient-Based method			
	Runtime		Objective value		Runtime		Objective value		Runtime	
	Stdv.		Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.
16	0.55E−2	5/2	164.53E−2	25.67E−2	0.16E−1	8.10E−3	**0.00**	0.00	**1.19E−2**	6.20E−03
17	10.94E−2	5/2	37.25E−2	0.01E−2	**2.89E−2**	3.80E−3	**0.18**	0.38E−2	1.02E−1	1.05E−02
18	0.07E−1	5/2	90.54E−2	23.01E−2	2.11E−2	1.36E−2	**0.00**	0.00	**0.82E−2**	3.60E−03
19	0.04E−1	5/2	8.41E−1	14.29E−2	**3.01E−2**	1.02E−2	**25.04E−9**	4.46E−09	3.02E−1	5.65E−02
20	9.54	10/200	1.56E−10	1.17E−11	152.97E−2	5.30E−2	**1.05E−13**	2.60E−15	**80.42E−2**	3.67E−01
21	0.07E−2	10/10	48600.64E−2	5514.29E−2	**27.64E−2**	2.58E−2	**9162.81E−2**	1342.76E−2	50.02E−2	2.21E−01
22	2.28E−1	10/2	21.69E−11	3.56E−11	0.88E−2	2.10E−3	**12.01E−14**	1.55E−15	**0.49E−2**	3.00E−04
Overall	**12/22**	–	1/22		8/22		**22/22**		**14/22**	

were allowed to fluctuate between zero and one; wherein, the smaller these gauges were, the more effective the boost mechanism was. These gauges were calculated for the above four algorithms and have been presented in Table A.2. Since, the calculated g_X^f did not exceed 0.5 in even a single case, the OBA was distinguished as a highly reliable algorithm. In addition, the median and mean values of g_X^f were considerably small for all of the boosted algorithms.

In conclusion, Table A.2 portrays the superb ability of the OBA to improve the quality of solutions. Since the applied methods were either gradient-based or simplex-based optimization techniques, they were expected to execute very quickly. Nevertheless, boosting each of these algorithms made them even faster, in many of the cases. While the medians of g_X^T for all of the boosted methods were smaller than 0.5, the mean calculated values for the boosted steepest descent and Nelder-Mead methods were 0.59 and 0.57, respectively. It is noteworthy that the performance of these two algorithms in finding the optimum solutions was improved considerably, such that their calculated mean values of g_X^f were 0.048 and 0.052, respectively, which implies that moderately lower runtimes for both of these two methods, before receiving a boost, was due to them becoming stuck in their local optima.

An approximate two-sample *t*-test was also conducted between each algorithm and its boosted version, where the statistical analysis method used was the same as that applied by Zhang, Luo, and Wang (2008). The following statistics were used for conducting the t-tests:

$$t_0 = \frac{\bar{y}_1 - \bar{y}_2}{\sqrt{S_1^2/n_1 + S_2^2/n_2}}, \quad df = \left\lfloor \left(\frac{\left(\frac{S_1^2}{n_1}\right)^2}{n_1} + \frac{\left(\frac{S_2^2/n_2}{S_1^2/n_1 + S_2^2/n_2}\right)^2}{n_2} \right)^{-1} \right\rfloor$$

$$\overset{n_1=n_2=n, SS_1=S_1^2, SS_2=S_2^2}{\rightarrow} t_0 = \frac{\bar{y}_1 - \bar{y}_2}{\sqrt{SS_1 + SS_2}}\sqrt{n}, \; df = \left\lfloor \frac{n(SS_1 + SS_2)^2}{SS_1^2 + SS_2^2} \right\rfloor \tag{12}$$

where y_1 and y_2, S_1 and S_2 denote the mean values and standard deviations of the results obtained by the traditional and boosted methods, respectively; n_1 and n_2 are the numbers of their independent runs; and *df* is the calculated degrees of freedom. The results, as depicted in Table A.3, were obtained at a confidence level of 0.95 ($\alpha = 0.05$), confirming the impressive ability of the boost mechanism to improve the performance of the tested optimization methods.

- **Meta-heuristics (PSO, DE, SA, VNS-SA, LSA and SOS)**

Several meta-heuristic methods, including the Particle Swarm Optimization (PSO), the Differential Evolution (DE), the Variable Neighborhood Search (VNS), the Simulated Annealing (SA) method, Lightening Search Algorithm (LSA) and Symbiotic Organisms Search (SOS) were coded via MATLAB, and the results are presented below. These algorithms were required to either converge, or reach a fixed tolerance ($\varepsilon < 10^{-5}$), before being terminated. A part of the optimization benchmark functions adopted from Table A.1 were applied to test the efficiency of the OBA while boosting these meta-heuristic optimization methods. The performances (including both objective errors and runtimes) of the PSO, DE, VNS-SA, LSA and SOS methods in solving benchmark functions were compared to those of the BPSO, BDE, BSA, BLSA and BSOS methods. Due to the random nature of the applied optimization methods, each benchmark function was solved by each algorithm for 60 times. The average objective errors and runtimes were calculated, and are reported in Tables 5 and 6.

The performances of PSO, DE, VNS and SA optimization methods in solving 50-variable instances are presented in Tables 5 and 6. The results demonstrate the effectiveness of the OBA in increasing the chances of finding optimum values when the PSO, DE or SA are receiving a boost. The VNS-SA was outperformed by the PSO and DE in several cases; whereas, the PSO and DE were outperformed by the VNS-SA in optimizing most of their benchmark functions. Therefore, in comparison to the PSO and the DE, the VNS-SA often reached better solutions.

Table 4
Nelder-Mead and BFGS methods versus their boosted versions.

Function number		Global optimum	Search domain/ dimension	Broyde-Fletcher-Goldfrab-Shanno Quasi-Newton (medium-scale) method				Boosted Broyde-Fletcher-Goldfrab-Shanno Quasi-Newton method			
				Objective value		Runtime		Objective value		Runtime	
				Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	
1	F01	0.00	100/2	305.98E+2	11363.73	1.41E−2	0.77E−2	**77.99E−13**	9.88E−13	**0.94E−2**	
2	F41	0.00	20/100	10.37E−2	0.27E−2	126.47E−2	44.68E−2	**9.17E−8**	2.09E−09	**120.24E−2**	
3	F49	0.00	10/100	16.78E−1	39.69E−2	146.84E−2	108.76E−2	**66.67E−2**	14.18E−2	**0.55E−2**	
4	F50	−180.13E−2	π/2	−31.79E−11	30.71E−2	**527.81E−7**	1.42E−5	**−180.13E−2**	0.00	10.24E−2	
5	F51	−468.76E−2	π/5	−164.03E−2	10.19E−1	2.92E−2	0.12E−2	**−355.69E−2**	28.62E−2	**2.17E−2**	
6	F52	−966.01E−2	π/10	−39.74E−1	55.75E−2	3.45E−2	0.27E−2	**−719.21E−2**	15.72E−2	**0.22E−2**	
7	F22	−386.27E−2	1/3	**−386.27E−2**	0.00	1.16E−2	0.02E−2	**−386.27E−2**	0.00	**0.17E−2**	
8	F25	−1014.99E−2	10/4	−263.05E−2	23.99E−1	2.14E−2	0.66E−2	**−1014.99E−2**	0.00	**1.23E−2**	
9	F26	−1039.99E−2	10/4	−276.59E−2	5.95E−2	1.25E−2	0.41E−2	**−1039.99E−2**	0.00	**0.76E−2**	
10	F27	−1053.19E−2	10/4	−287.11E−2	0.42	2.72E−2	0.82E−2	**−1053.19E−2**	0.00	**2.26E−2**	
11	F28	−3322.36E−3	10/6	0.00	6.41E−2	1.81E−2	0.48E−2	**−3322.36E−3**	0.00	**1.73E−2**	
12	F39	−3456.00	500/2	328.91E−13	868.03	**1.55E−2**	0.81E−2	**−3456.00**	0.00	40.29E−2	
13	F40	0.00	10/10	103.38E−12	3.09E−11	1.47E−2	0.14E−2	**376.11E−17**	1.55E−15	**0.91E−2**	
14	F45	−4189.83E−1	5/500	−20174.42E−2	3908.24E−2	**153.94E−2**	80.19E−2	**−32422.56E−2**	281.49E−2	1000.23E−2	
15	F46	−50.00	36/6	**−50.00**	0.00	1.19E−2	0.16E−2	**−50.00**	0.00	**3.42E−3**	
16	F47	−210.00	100/10	**−210.00**	0.00	2.08E−2	0.13E−2	**−210.00**	0.00	**0.78E−2**	
17	F32	0.00	500/10	1383.83E−2	273.39E−2	119.13E−2	8.12E−2	**1333.99E−2**	484.96E−2	**20.71E−2**	
18	F33	0.00	30/20	19910.84E−2	3322.17E−2	294.65E−2	14.01E−2	**414.57E−13**	1.64E−11	**120.47E−2**	
19	F37	0.00	10/5	23105.78E−2	8057.28E−2	9.82E−2	4.26E−2	**12349.80E−2**	3201.67E−2	**8.16E−2**	
20	F48	0.00	10/80	15.61E−11	8.16E−11	152.97E−2	44.89E−2	**10.49E−14**	1.45E−14	**80.42E−2**	
21	F07	−783.32E−1	5/2	−6419.56E−2	567.42E−2	**1.91E−2**	0.89E−2	**−7833.23E−2**	0.00	2.94E−2	
22	F11	0.00	π/2	17.75E−2	1.43E−2	**1.03E−2**	0.16E−2	**244.25E−17**	1.35E−16	20.85E−2	
23	F24	0.00	2π/1000	392.76E−58	4.92E−57	1.69E−2	0.09E−2	**0.00**	0.00	**1.17E−2**	
Overall				3/23		5/23		**23/23**		**18/23**	

Function number	Boosted Broyde-Fletcher-Goldfrab-Shanno Quasi-Newton method	R/d	Nelder-Mead method				Boosted Nelder-Mead method			
	Runtime		Objective value		Runtime		Objective value		Runtime	
	Stdv.		Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.
1	0.49E−2	100/2	48.98	919.85E−2	**3.75E−2**	0.56E−2	**1.62E−9**	1.18E−10	0.1337	8.83E−2
2	16.19E−2	20/100	1.84E+6	14810.61E+2	101.02	4306.92E−2	**3.75E−4**	0.01E−2	**37.02**	451.97E−2
3	0.11E−2	10/100	1.08E+5	12134.33	28.37	468.93E−2	**6.67E−1**	19.42E−2	**18.91**	248.21E−2
4	0.33E−2	π/2	−41.31E−2	55.04E−2	14.68E−2	6.84E−2	**−180.13E−2**	0.00	**5.03E−2**	0.73E−2
5	0.14E−2	π/5	−177.73E−2	9.97E−1	**1.88E−2**	0.43E−2	**−468.76E−2**	0.00	21.35E−1	14.21E−2
6	3.90E−05	π/10	−23.81E−1	89.05E−2	**3.55E−2**	0.35E−2	**−966.01E−2**	0.00	2.83	35.12E−2
7	0.01E−2	1/3	−100.08E−2	95.77E−2	10.63E−2	1.98E−2	**−386.27E−2**	0.00	**4.29E−2**	0.27E−2
8	0.19E−2	10/4	−268.29E−2	195.37E−2	1.91E−2	0.75E−2	**−1014.99E−2**	0.00	**1.36E−2**	0.71E−2
9	0.05E−2	10/4	−275.19E−2	287.11E−2	**0.22E−11**	0.23E−2	**−1039.99E−2**	0.00	23.52E−2	5.13E−2
10	1.23E−2	10/4	−242.17E−2	41.42E−2	1.97E−2	1.05E−2	**−1053.19E−2**	0.00	**1.31E−2**	0.24E−2
11	0.76E−2	10/6	0.00	117.79E−2	**2.41E−2**	0.31E−2	**−3322.36E−3**	0.00	0.07	3.11E−2

(*continued on next page*)

Table 4 (*continued*)

Function number	Boosted Broyde-Fletcher-Goldfrab-Shanno Quasi-Newton method	R/d	Nelder-Mead method				Boosted Nelder-Mead method			
	Runtime		Objective value		Runtime		Objective value		Runtime	
	Stdv.		Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.
12	20.32E − 2	500/2	14.04E − 8	2083.02E − 1	**0.08**	2.29E − 2	**− 3456.00**	0.00	10.03E − 1	57.27E − 2
13	0.09E − 2	10/10	272.51E − 1	161.05E − 2	14.97E − 2	6.81E − 2	**30.32E − 6**	1.92E − 08	**12.04E − 2**	2.19E − 2
14	117.52E − 2	5/500	− 2057.72E − 1	3492.87E − 2	**191.61E − 2**	15.52E − 2	**− 3795.03E − 1**	1423.63	450.44E − 2	103.69E − 2
15	0.02E − 2	36/6	**− 50.00**	0.00	**4.01E − 2**	0.83E − 2	**− 50.00**	0	6.95E − 2	0.43E − 2
16	0.76E − 2	100/10	− 9406.56E − 2	17.75	12.01E − 2	0.22E − 1	**− 210.00**	0	**11.23E − 2**	5.54E − 2
17	0.23E − 1	10/200	1422.09E − 2	401.42E − 2	1472.99E − 2	542.98E − 2	**1317.05E − 2**	106.34E − 2	**1453.08E − 2**	317.27E − 2
18	3.14E − 2	30/20	49667.00	11533.64	**18.72E − 1**	48.36E − 2	**2.88E − 8**	8.50E − 09	9.83	11.21E − 2
19	2.08E − 2	10/5	16632.16E − 2	7103.09E − 2	**95.47E − 2**	23.87E − 2	**896.37E − 2**	227.77E − 2	370.26E − 2	171.48E − 2
20	30.67E − 2	10/80	1.43E + 3	62598.67E − 2	31.74	524.59E − 2	**40.11E − 2**	1.27E − 2	**31.23**	441.47E − 2
21	1.62E − 2	5/2	− 5005.89E − 2	416.64E − 2	7.19E − 2	1.92E − 2	**− 783.32E − 1**	0.00	**4.31E − 2**	1.03E − 2
22	0.33E − 2	π/2	68.51E − 2	22.56E − 2	28.64E − 2	14.25E − 2	**2.12E − 8**	3.63E − 09	**4.73E − 2**	2.83E − 2
23	0.06E − 2	2π/1000	1.88E − 36	2.64E − 37	15.14E − 2	5.15E − 2	**2.27E − 60**	1.17E − 61	**11.49E − 2**	1.09E − 2
Overall	**18/23**	–	1/23		10/23		**23/23**		**13/23**	

The PSO, DE and VNS-SA reached the predetermined fixed tolerances in 3, 4 and 6 out of the 11 aforementioned cases, respectively; and as shown in Table 5, boosting these algorithms made them capable of reaching their fixed tolerances when optimizing all of their benchmark functions. Nevertheless, as it has also been proven mathematically in Section 2.3, it is clear that the quality of the optimum solutions found by these three algorithms would have been improved in each case, after receiving a boost, or at least remained unchanged.

The mean runtimes consumed by these algorithms and their boosted versions to converge, or to reach the fixed tolerance levels, are presented in Table 6. In addition to the improvements made in the obtained objective values, the required runtimes were also substantially diminished. In many cases, the VNS enabled the SA to find better solutions than the PSO and the DE, while consuming less computation times; but in almost all cases, both the objective errors and required runtimes for the SA, after receiving a boost, were much smaller than those of the VNS-SA.

Most recently, number of meta-heuristic optimization methods has increased rapidly due to the growing applications of such algorithms for solving business and academic problems. However the key purpose of this section is to measure OBA's impact on a considerably wide range of optimization methods especially the meta-heuristic algorithms. In this regard, several most-frequently meta-heuristic optimization algorithms such as PSO, DE and SA are chosen in addition to two recently developed and significantly powerful meta-heuristic optimization methods that are called LSA (Shareef et al., 2015) and SOS (Cheng & Prayogo, 2014). The results of boosting these two algorithms in terms of both obtained objective values and execution times, are reported through Tables 7 and 8.

The applied population size for both LSA and SOS is 50 whereas channel time for LSA is assumed to be 10. Although, these two algorithms are showing very good performance even in their traditional form, significant improvements in both obtained objective value and required runtime are observed as the result of them receiving a boost. BLSA accomplished to find better objective values in 3 cases out of 10, while consuming considerably less execution times in 7 cases. On the other hand, boosting SOS resulted in better objective values being found in 2 cases as well as consuming less computation time in 9 cases out of 10. As shown in Tables 5–8, boosting meta-heuristic optimization algorithms not only usually enables them to find better solutions, but also decreases their required runtimes by increasing their respective intelligences. Remainder of the parameters of the boosted and traditional meta-heuristic algorithms are presented in Tables A.4 and A.5.

In order to measure the effects of boosting algorithms, on their sensitivities to problem sizes, a comparison was made between the PSO and the Boosted PSO (BPSO) in their abilities to optimize the Ackley function. This function has many widespread local minima, but it has only one global minimum ($x_i = 0$). The search domain, while applying the PSO/BPSO, was 10 (R = 10), and the results that were obtained while optimizing the Ackley function with 50 to 400 variables (d = 50/200/400), are shown in Fig. 14.

As shown in Fig. 14, the BPSO outperformed the PSO when optimizing the Ackley function with 50, 200 and 400 variables; and for virtually any amount of time, the optimum solutions found by the PSO were surpassed by those of the BPSO. As time went on, the gap between the BPSO and the PSO increased, which confirmed the outstanding ability of BPSO to solve enormous business optimization problems.

Next, the DE was boosted to optimize the Rosenbrock benchmark function, where the search domain was assumed to be 30 (R = 30). The overall function, $\sqrt[10]{f(\vec{x})}$, is plotted in Fig. 15, comparing the performance of the DE with the Boosted DE (BDE). Although increasing the number of variables affected the performance of both the DE and the BDE, the boosted version found better solutions compared to those of the traditional DE, while consuming the same time.

The benchmark function that was applied to investigate the

Table 5
Mean errors obtained by aforementioned optimization methods before and after receiving a boost.

Number	Benchmark function	Global optimum	Search domain/dimension	PSO		BPSO		DE		BDE		VNS-SA		BSA	
				Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.
1	F32	0.00	10/50	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–
2	F33	0.00	30/50	163.74	21967.49E−3	$< \mathbf{10^{-5}}$	–	21.45	9604.33E−3	$< \mathbf{10^{-5}}$	–	27.65	498.65E−2	$< \mathbf{10^{-5}}$	–
3	F34	0.00	50/50	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	22.61	7048.98E−3	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–
4	F35	0.00	100/50	0.29	120.97E−3	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–
5	F36	0.00	1.28/50	2.58	1037.66E−3	$< \mathbf{10^{-5}}$	–	13.62E−1	187.13E−3	$< \mathbf{10^{-5}}$	–	9.07E−1	155.72E−3	$< \mathbf{10^{-5}}$	–
6	F37	0.00	5.12/50	115.73	15290.72E−3	$< \mathbf{10^{-5}}$	–	63.82	21340.86E−3	$< \mathbf{10^{-5}}$	–	34.04	12000.13E−3	$< \mathbf{10^{-5}}$	–
7	F38	0.00	50/50	1.68E−1	46.13E−3	$< \mathbf{10^{-5}}$	–	4.03	209.04E−3	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–
8	F41	0.00	20/50	4.64	1452.96E−3	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	0.06E−2	0.27E−3	$< \mathbf{10^{-5}}$	–
9	F42	0.00	100/50	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	32.93	6626.61E−3	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–
10	F43	0.00	10/50	0.03	2.96E−3	$< \mathbf{10^{-5}}$	–	4.67	485.49E−3	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–
11	F44	0.00	100/50	0.31	24.84E−3	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	$< \mathbf{10^{-5}}$	–	0.14	59.08E−3	$< \mathbf{10^{-5}}$	–
Overall				3/11		**11/11**		4/11		**11/11**		6/11		**11/11**	

Table 6
Mean runtimes obtained by aforementioned optimization methods before and after receiving a boost.

Number	Benchmark	Search domain/dimension	PSO		BPSO		*t*-test result	DE		BDE		*t*-test result	VNS-SA		BSA		*t*-test result
			Mean	Stdv.	Mean	Stdv.		Mean	Stdv.	Mean	Stdv.		Mean	Stdv.	Mean	Stdv.	
1	F32	10/50	161.95	112.85E−1	**13.91**	38.96E−1	BPSO	120.99	234.02E−1	**32.97**	7.48E−1	BDE	69.36	75.57E−1	**34.92**	53.33E−1	BSA
2	F33	30/50	639.79	103.27	**127.75**	119.67E−1	BPSO	647.83	446.34E−1	**74.38**	33.93E−1	BDE	431.96	616.37E−1	**134.21**	393.98E−1	BSA
3	F34	50/50	198.46	443.85E−1	**78.25**	104.33E−1	BPSO	163.65	370.36E−1	**66.58**	135.38E−1	BDE	85.48	35.31E−1	**27.92**	44.84E−1	BSA
4	F35	100/50	476.52	323.59E−1	**97.96**	6.18	BPSO	432.54	81.97E−1	**100.73**	75.83E−1	BDE	283.75	1141.37E−1	**93.62**	154.34E−1	BSA
5	F36	1.28/50	503.84	118.65	**102.91**	270.02E−1	BPSO	387.93	493.69E−1	**110.85**	265.25E−1	BDE	327.63	559.42E−1	**84.38**	18.96	BSA
6	F37	5.12/50	436.48	76.26	**90.73**	19.92	BPSO	376.27	199.18E−1	**121.68**	80.16E−1	BDE	211.35	471.87E−1	**73.93**	257.88E−1	BSA
7	F38	50/50	158.85	344.67E−1	**60.75**	70.36E−1	BPSO	184.55	563.01E−1	**93.71**	82.98E−1	BDE	61.03	72.54E−1	**59.19**	3.79	–
8	F41	20/50	159.87	31.08E−1	**158.87**	15.78E−1	–	172.26	397.22E−1	**56.51**	0.97E+1	BDE	**82.68**	31.15E−1	8.53E+1	26.49E−1	VNS-SA
9	F42	100/50	268.48	625.67E−1	**99.44**	105.02E−1	BPSO	259.44	365.46E−1	**129.06**	30.49E−1	BDE	210.05	194.09E−1	**97.44**	34.61E−1	BSA
10	F43	10/50	183.72	424.52E−1	**76.37**	141.89E−1	BPSO	**161.88**	68.53E−1	172.26	2.23	DE	159.73	243.41E−1	**75.12**	10.32	BSA
11	F44	100/50	168.38	589.37E−1	**64.84**	92.49E−1	BPSO	207.54	557.86E−1	**84.36**	134.36E−1	BDE	185.79	18.57	**64.53**	13.53	BSA
Overall			0/11		**11/11**		10 > 0	1/11		**10/11**		10 > 1	1/11		**10/11**		9 > 1

Table 7
Mean errors and runtimes obtained by LSA before and after receiving a boost.

Number	Benchmark function	Global optimum	Search domain/ dimension	LSA				BLSA			
				Error		Runtime		Error		Runtime	
				Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.
1	F22	−3.862782	1/3	$< \mathbf{10^{-5}}$	–	1.09043	0.131895	$< \mathbf{10^{-5}}$	–	**0.859804**	0.11092
2	F28	−3.32236	10/6	$< \mathbf{10^{-5}}$	–	**3.245191**	0.427905	$< \mathbf{10^{-5}}$	–	3.401757	0.535756
3	F32	0	10/50	$< \mathbf{10^{-5}}$	–	32.18518	3.19262	$< \mathbf{10^{-5}}$	–	**16.59028**	3.120865
4	F33	0	30/50	1.478477	0.743848	**481.141**	33.53661	$< \mathbf{10^{-5}}$	–	519.5433	56.42405
5	F34	0	50/50	$< \mathbf{10^{-5}}$	–	55.82483	4.917646	$< \mathbf{10^{-5}}$	–	**32.48664**	2.798185
6	F36	0	1.28/50	$< \mathbf{10^{-5}}$	–	353.7258	26.15961	$< \mathbf{10^{-5}}$	–	**172.2243**	32.76083
7	F37	0	5.12/50	0.00753143	0.0022373	240.0884	30.12969	$< \mathbf{10^{-5}}$	–	**174.909**	17.09678
8	F38	0	50/50	2.672 E−4	2.2946E−4	**58.11083**	5.187831	$< \mathbf{10^{-5}}$	–	64.4557	12.60919
9	F42	0	100/50	$< \mathbf{10^{-5}}$	–	203.4646	30.46225	$< \mathbf{10^{-5}}$	–	**128.7792**	15.47771
10	F44	0	100/50	$< \mathbf{10^{-5}}$	–	212.5929	32.75627	$< \mathbf{10^{-5}}$	–	**124.3076**	10.66708
Overall				7/10		3/10		**10/10**		**7/10**	

performance of the hybrid VNS-SA versus the BSA was the Sphere function. The search domain, while optimizing the Sphere function, was 20 (R = 20), and Sphere functions with 100, 250 and 500 variables (d = 100/250/500) were each solved using both the VNS-SA and BSA, with results shown in Fig. 16. Both the VNS-SA and BSA showed promising results in optimizing the Sphere function, and since some important differences cannot be observed in the figures, the final optimum values that were found by both algorithms and their runtimes are reported in Table 9. As shown in this table, unlike the VNS-SA, the BSA reached fixed tolerances of 10–5 in 2 out of the 3 cases; and the optimum value found by the BSA in the third case, with 500 variables, was considerably smaller than the one found by the VNS-SA. In addition, the VNS-SA took 48.4% more time than the BSA to converge, when solving the Sphere function with 500 variables.

3.3. Constrained non-linear benchmarks

Twelve constrained and continuous benchmark optimization problems, with a variety of 2 to 100 variables, and 1 to 9 constraints were adopted from CEC 2010 and the previous works of Deb (2000), Lee and Geem (2005), and Oftadeh, Mahjoob, and Shariatpanahi (2010). These benchmark problems are presented in Table A.6, in the following generalized form:

$$\begin{aligned} &\min \quad f(\vec{x}) \\ &s.\ t. \\ &g_i(\vec{x}) = 0 \quad or \quad g_i(\vec{x}) \leqslant 0 \end{aligned} \tag{13}$$

For constrained problems, the constrained OBA (COBA) uses $U(\vec{x})$ instead of the conventional objective function $f(\vec{x})$. The objective value and the solutions' deviation from the feasible space are minimized simultaneously by minimizing$U(\vec{x})$:

$$\begin{aligned} \min \quad & U(\vec{x}) = f(\vec{x}) + bigm \times \textstyle\sum_i V_i(\vec{x}) \qquad bigm \to \infty \\ & g_i(\vec{x}) = 0 \Rightarrow V_i(\vec{x}) = |g_i(\vec{x})| \\ & g_i(\vec{x}) \leqslant 0 \Rightarrow V_i(\vec{x}) = 0.5 \times (g_i(\vec{x}) + |g_i(\vec{x})|) \end{aligned} \tag{14}$$

The desired goal is to find the solution $\vec{x}^*$ with the smallest possible value of $U(\vec{x}^*)$, where $\vec{x}^*$ is the global optimum solution among the feasible solutions of the constrained problem.

- **Active Set, Sequential Quadratic Programming (SQP) and Interior Point methods**

As in the case of many other non-linear optimization techniques, the Active Set Method uses solutions of the Karush-Kuhn-Tucker (KKT) equations. This algorithm attempts to compute Lagrange multipliers directly; wherein, SQP effectively acts similarly to the Active Set method. SQP generates a Quadratic Programming (QP) sub-problem using an approximation of the Hessian of the Lagrangian function made by a Quasi-Newton updating method. Then, a line search procedure takes advantage of the QP sub-problem's solution to form a search direction (Fletcher, 1987; Gill, Murray, & Wright, 1981; Hock & Schittkowski, 1983; Powell, 1983). The Interior Point method is also a constrained minimization technique, which solves a sequence of approximate minimization problems. This iterative method uses one of the following two types of steps at each iteration: (1) A Newton step, in which the algorithm tries to solve KKT equations; or (2) A conjugate

Table 8
Mean errors and runtimes obtained by SOS before and after receiving a boost.

Number	Benchmark function	Global optimum	Search domain/dimension	SOS				BSOS			
				Error		Runtime		Error		Runtime	
				Mean	Stdv.	Mean	Stdv.	Mean	Stdv.	Mean	Stdv.
1	F06	−1.03163	5/2	$< \mathbf{10^{-5}}$	–	1.20E+01	9.32E−01	$< \mathbf{10^{-5}}$	–	**5.51E+00**	9.90E−01
2	F14	0	10/2	−9.65991	0.00133	**8.66E+01**	1.56E+01	$< \mathbf{10^{-5}}$	–	10.38E+01	1.97E+01
3	F21	0	10/2	$< \mathbf{10^{-5}}$	–	3.01E+02	5.62E+01	$< \mathbf{10^{-5}}$	–	**2.22E+02**	3.12E+01
4	F23	0	10/4	$< \mathbf{10^{-5}}$	–	3.54E+01	6.65E+00	$< \mathbf{10^{-5}}$	–	**1.72E+01**	1.60E+00
5	F35	0	600/30	$< \mathbf{10^{-5}}$	–	8.67E+01	1.58E+01	$< \mathbf{10^{-5}}$	–	**3.95E+01**	7.88E+00
6	F40	0	10/30	$< \mathbf{10^{-5}}$	–	2.23E+01	2.81E+00	$< \mathbf{10^{-5}}$	–	**1.98E+01**	2.23E+00
7	F49	0	10/30	$< \mathbf{10^{-5}}$	–	2.10E+02	2.08E+01	$< \mathbf{10^{-5}}$	–	**1.78E+02**	1.68E+01
8	F50	−1.8013	π/2	$< \mathbf{10^{-5}}$	–	3.63E+00	4.48E−01	$< \mathbf{10^{-5}}$	–	**2.23E+00**	3.61E−01
9	F51	−4.6876	π/5	$< \mathbf{10^{-5}}$	–	4.29E+00	7.00E−01	$< \mathbf{10^{-5}}$	–	**3.69E+00**	4.36E−01
10	F52	−9.6601	π/10	0.028641	0.109758	6.37E+00	1.16E+00	$< \mathbf{10^{-5}}$	–	**4.86E+00**	9.50E−01
Overall				8/10		1/10		**10/10**		**9/10**	

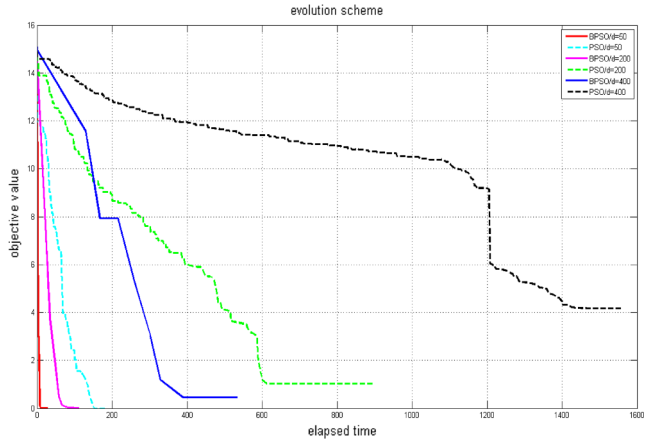


Fig. 14. . PSO and BPSO results for Ackley function.

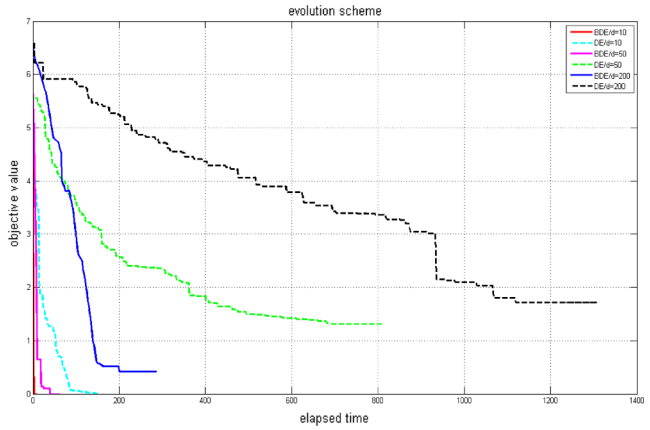


Fig. 15. DE and BDE results for Rosenbrock function.

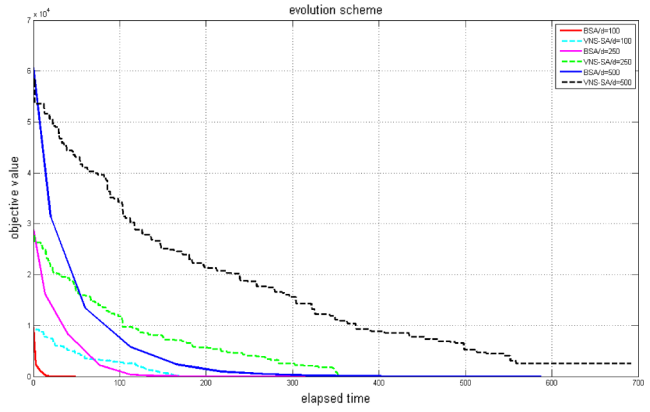


Fig. 16. VNS-SA and BSA results for Sphere function.

gradient step, in which a trust region approach is applied (Byrd, Gilbert, & Nocedal, 2000; Byrd, Hribar, & Nocedal, 1999; Waltz, Morales, Nocedal, & Orban, 2006).

Once again, for this study, MATLAB codes were provided by the MATLAB 2012a optimization toolbox, Optimtool, and constrained benchmarks, as presented in Table A.6, were solved by these three constrained optimization techniques and their boosted versions. The comparative results and median values are provided in Table A.7, where each problem was solved a total of 60 times.

The best results found by each algorithm and/or its boosted version are written in bold. The results revealed the excellent ability of the COBA to improve the quality of the solutions, in terms of both their objective values and feasibilities, as well as to decrease their required runtimes. In many cases, applying COBA resulted in significant improvements in the quality of the final solutions, while it enabled the Interior Point method to find better values for $U(\vec{x})$ and $f(\vec{x})$ in 11 and 9 out of the 12 cases, respectively. The $U(\vec{x})$ and $f(\vec{x})$values found by the methods of SQP and Active Set were also improved after receiving a boost in 9, 6, 11 and 4 out of the 12 cases, respectively. The boosted SQP and Active Set methods were also able to diminish the deviations from the feasible solution spaces for 13 and 20 out of the total 47 constraints; while diminishing the deviations also made them capable of finding 4 and 2 more final feasible solutions out of the 12 tested benchmarks, respectively. Due to the low order of these three methods, they were expected to have had considerably lower runtimes — regardless of whether or not they had managed to find an optimal and feasible solution. However, in many cases, boosting these methods made them capable of finding better solutions in even shorter runtimes. Fig. 17 is presented for further comparison.

Table A.7, and Fig. 17 are very helpful for depicting how the OBA works. The Interior Point method satisfied 42 out of the 47 constraints, and found fully feasible solutions for 10 out of the 12 cases. Thus, the performance of the Interior Point method in finding feasible solutions and satisfying constraints was found to be far more promising than those of the SQP and Active Set methods. However, the OBA still improved the Interior Point method to find better solutions, with much better objective values; while the feasibility of the final solutions increased too. In fact, the OBA boosted the Interior Point method to augment its strengths (by finding more feasible solutions) and reduce its weaknesses (by finding better objective values). Although each of these three constrained optimization methods have strengths and weaknesses of their own, boosting these algorithms improved them to an extent that they were able to achieve better performances regarding both their strengths and their weaknesses. For instance, while the major deficiency of the SQP and the Active Set methods in solving the proposed benchmarks turned out to be finding feasible solutions, the COBA improved their performances considerably, and increased their chances of finding the most optimum and feasible solutions. In addition to the noticeable improvement made in the quality of final solutions, boosting Interior Point, SQP and Active Set methods led into saving computation time in 8, 7 and 7 cases out of 12. As it is highlighted in Table A.7, boosted algorithms become more dominant compared to their traditional forms as the search space dimensionality grows. Better performance of Interior Point, SQP and Active set methods in terms of both optimality and execution time after receiving a boost while solving a constrained optimization problem with 100 variables exemplifies this fact. It should be noted that the parameter value intervals for the boosted constrained optimization methods were the same as those that were used for the boosted iterative gradient-based unconstrained methods.

4. OBA applications for solving real-world engineering problems

Three real-world engineering design optimization problems were applied, next, to observe the performance of the boosted algorithms in facing the complexities of real-world engineering problems.

- **A real-world bi-objective centrifugal pump design engineering problem**

Nourbakhsh et al. introduced the optimization of centrifugal pumps as a bi-objective optimization problem (Nourbakhsh, Safikhani, & Derakhshan, 2011). They attempted to simultaneously maximize the efficiency (η), and minimize the required Net Positive Suction Head (NPSHr) of centrifugal pumps. GMDH-type neural networks led them to the following polynomial representations for η and NPSHr:

Table 9
The effect of boosting several meta-heuristic algorithms on their sensitivity to the problem size.

Solver/Solver/Problem	Global optimum	Search domain/dimension	Traditional optimization method		Boosted optimization method	
			Objective value	Runtime (s)	Objective value	Runtime (s)
PSO/BPSO/Ackley	0.00	10/50	$<10^{-5}$	148.67	$<10^{-5}$	8.93
	0.00	10/200	1.03	611.78	$<10^{-5}$	111.61
	0.00	10/400	4.17	1465.62	0.46	388.86
DE/BDE/Rosenbrock	0.00	30/10	$<10^{-5}$	161.84	$<10^{-5}$	6.93
	0.00	30/50	15.66	688.06	$<10^{-5}$	39.18
	0.00	30/200	217.31	1119.28	0.16E−3	199.95
VNS-SA/BSA/Sphere function	0.00	50/100	8.74	171.94	$<10^{-5}$	48.51
	0.00	50/250	53.75	355.46	$<10^{-5}$	112.53
	0.00	50/500	2611.93	558.39	709.12E−2	376.22

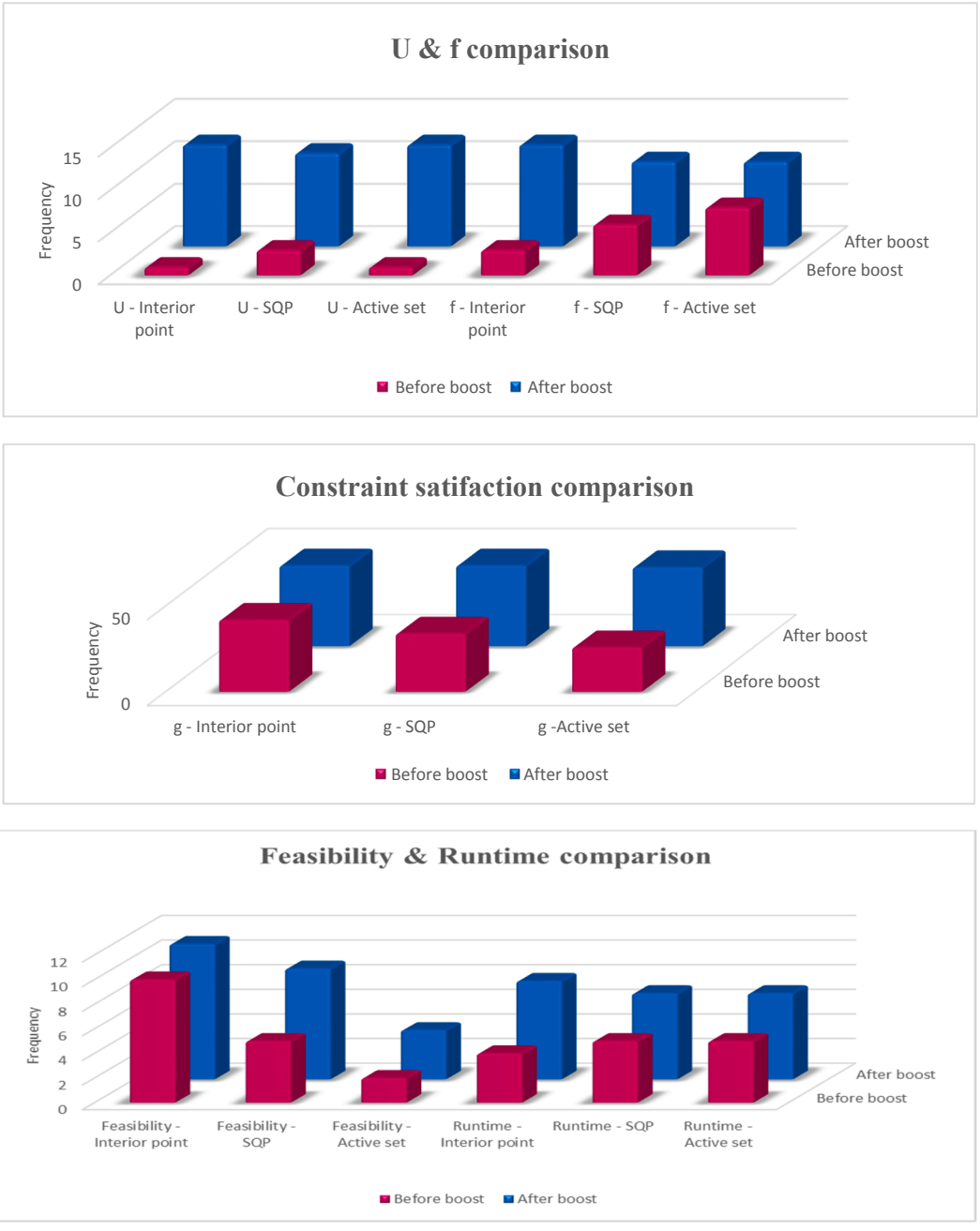


Fig. 17. Performance comparison for before and after receiving a boost while solving constrained benchmarks.

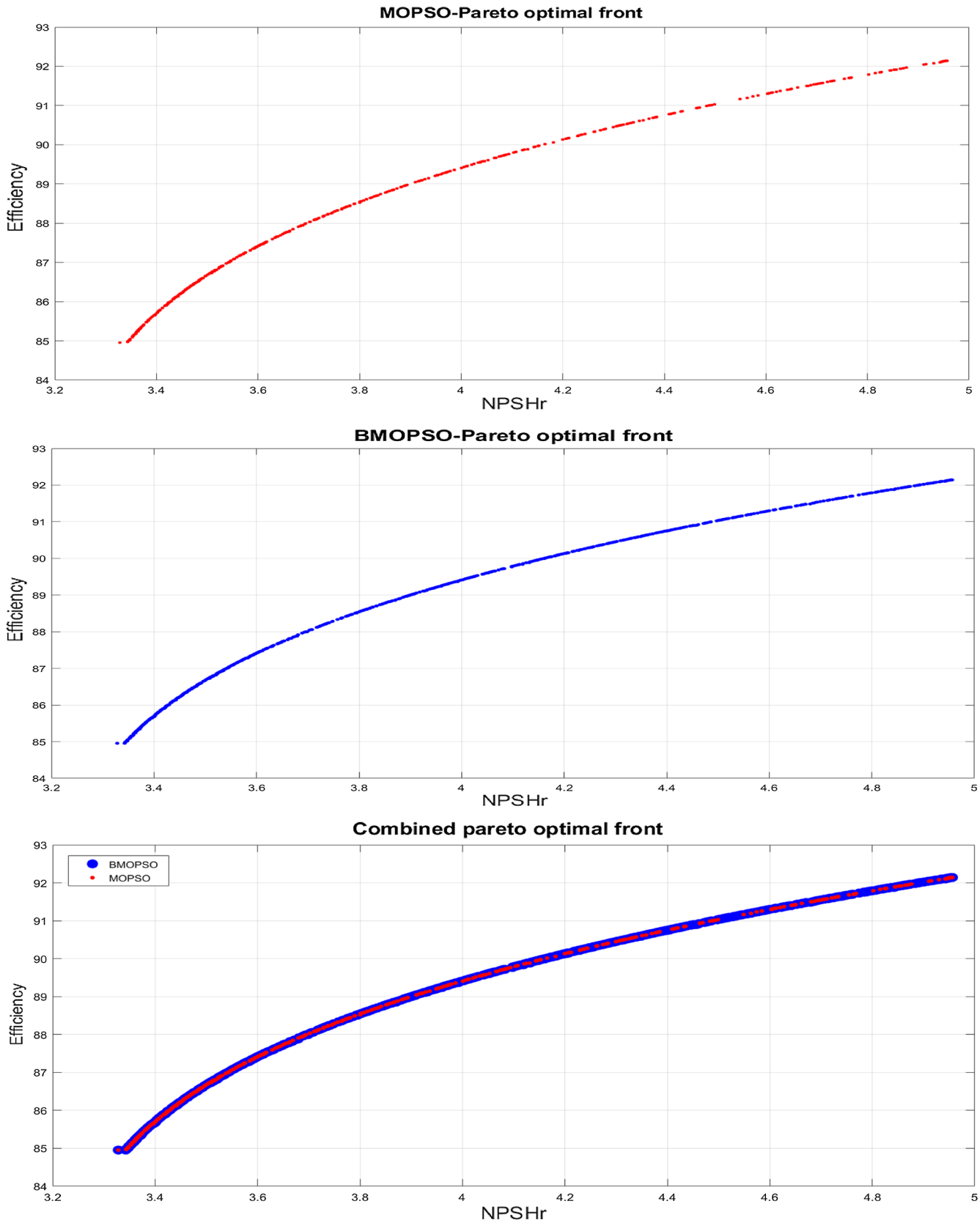
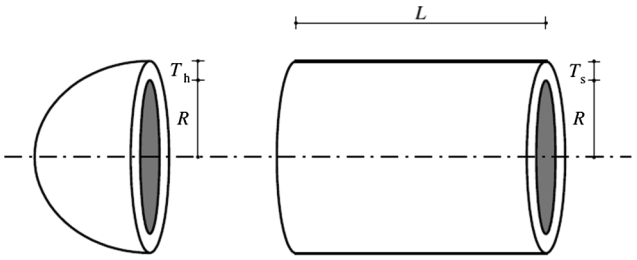


Fig. 18. Pareto optimal fronts for the bi-objective centrifuge pump design engineering problem.



T_s: Shell thickness, T_h: Spherical head thickness
R: Radius of cylindrical shell, L: Shell length,

Fig. 19. Pressure vessel design engineering optimization problem.

$$
\begin{aligned}
Y_1 &= 0.476 - 0.33\beta_{1Hub} + 2.027\beta_{1Shroud} + 0.014\beta_{1Hub}^2 - 0.013\beta_{1Shroud}^2 \\
&\quad + 0.0001\beta_{1Hub}\beta_{1Shroud} \\
Y_2 &= 20.3595 + 1.1797\gamma_{\text{mid}} + 0.5391\beta_2 + 0.00787\gamma_{mid}^2 - 0.00397\beta_2^2 \\
&\quad + 0.00115\gamma_{\text{mid}}\beta_2 \\
Y_3 &= -17.93 + 2.01\beta_{1Shroud} - 0.5805\beta_2 - 0.013\beta_{1Shroud}^2 - 0.00397\beta_2^2 \\
&\quad + 0.0001\beta_{1Shroud}\beta_2 \\
Y_4 &= 37.03 + 1.228\gamma_{\text{mid}} - 0.352\beta_{1Hub} - 0.0078\gamma_{mid}^2 + 0.0142\beta_{1Hub}^2 \\
&\quad + 0.00062\gamma_{\text{mid}}\beta_{1Hub} \\
Y_5 &= 60.5535 - 0.66962Y_1 - 0.91212Y_2 + 0.004299Y_1^2 + 0.005978Y_2^2 \\
&\quad + 0.012869Y_1Y_2 \\
Y_6 &= 57.0403 - 0.52137Y_3 - 0.97334Y_4 + 0.00336Y_3^2 + 0.0063741Y_4^2 \\
&\quad + 0.0128Y_3Y_4 \\
\eta &= 0.68350 - 3.2012Y_5 + 4.39817Y_6 - 2.330201Y_5^2 - 2.37611Y_6^2 \\
&\quad + 4.706481Y_5Y_6
\end{aligned} \tag{15}
$$

$$
\begin{aligned}
Y_1' &= -1.62 - 0.014\beta_{1Hub} + 0.16\beta_{1Shroud} + 0.005\beta_{1Hub}^2 - 0.0010\beta_{1Shroud}^2 \\
&\quad + 2.2 \times 10^{-6}\beta_{1Hub}\beta_{1Shroud} \\
Y_2' &= 3.426 - 0.0152\beta_{1Hub} + 0.0177\beta_2 + 0.0005\beta_{1Hub}^2 + 3.109 \times 10^{-12}\beta_2^2 \\
&\quad + 1.22 \times 10^{-5}\beta_2\beta_{1Hub} \\
Y_3' &= 6.175 - 12.92\gamma_{\text{mid}} - 0.01609\beta_{1Hub} + 0.001\gamma_{mid}^2 + 0.0005\beta_{1Hub}^2 \\
&\quad + 2.83 \times 10^{-5}\gamma_{\text{mid}}\beta_{1Hub} \\
Y_4' &= -2.47 + 0.159\beta_{1Shroud} + 0.017\beta_2 + 0.0010\beta_{1Shroud}^2 + 1.27 \times 10^{-11}\beta_2^2 \\
&\quad + 1.35 \times 10^{-5}\beta_{1Shroud}\beta_2 \\
Y_5' &= 7.5491 - 3.5231Y_1' - 0.4288Y_2' + 0.46290Y_1'^2 + 0.019501Y_2'^2 \\
&\quad + 0.060303Y_1'Y_2' \\
Y_6' &= 5.9118 - 1.963Y_3' - 0.6936Y_4' + 0.21291308Y_3'^2 + 0.0801906Y_4^{v2} \\
&\quad + 0.2316076Y_3'Y_4' \\
NPSHr &= -0.3809 - 0.0652Y_5' + 1.217Y_6' - 0.0704Y_5'^2 + 0.030280Y_6'^2 \\
&\quad - 0.1160804Y_5'Y_6'
\end{aligned} \tag{16}
$$

where $\beta_{1Shroud}$, β_{1Hub}, γ_{mid} and β_2 are design variables. Consequently, the following bi-objective continuous and unconstrained optimization problem was introduced by Nourbakhsh et al. (2011), as a centrifugal pump engineering design problem:

Maximum η
Minimum NPSHr

Table 10
Interior point, SQP and active set methods versus their boosted versions in solving pressure vessel design optimization problem.

Problem number	Results	Interior point method		Boosted interior point method	SQP	Boosted SQP	Active set		Boosted active set
Pressure vessel design	Objective	$U(\vec{x})$		1191324.75E−1	**7436381.95E−3**	1191323.59E−1	**7388523.15E−3**	1191323.59E−1	**7489113.96E−3**
		$f(\vec{x})$		**6632470.86E−3**	7436381.95E−3	**6632471.09E−3**	7388523.15E−3	**6632471.09E−3**	7489113.96E−3
	Constraints	$g(\vec{x})$		3750011.63E−8	**−1292538.83E−8**	3.75E−2	**−17461.69E−6**	3.75E−2	**−2842615.29E−8**
				−1876937.25E−8	**−7530094.32E−8**	**−1876943005 18.13E−11**	**−77543.24E−6**	**−1876943.00E−8**	**−8296297.92E−8**
				−4761547.75E−8	**−4559316.40E−2**	**−2328306.44E−16**	**−3063840.01E−2**	2328306.44E−16-	**−4487490.73E−2**
				−1889343.65E	**−1881392.90E−4**	**−1889343.35E−4**	**−1882158.32E−4**	**1889343.35E−4**	**−1834753.98E−4**
				−43.75E−23.75E−2	**−0.25E−1−0.25E−1**	3.75E−023.75E−02	**−0.25E−1−0.25E−1**	3.75E−23.75E−2	**−0.25E−1−0.25E−1**
	Feasibility			Infeasible	**Feasible**	Infeasible	**Feasible**	Infeasible	**Feasible**
	Decision variables	$\vec{x}$		106.25E−256.25E−2	11.25E−1 6.25E−1	106.25E−2	11.25E−1 6.25E−1	106.25E−256.25E−2	11.25E−1 6.25E−1
				5699482.47E−5	5762044.62E−5	56.25E−2 5699481.86E−5	5738540.43E−5	5699481.86E−5	5681729.78E−5
				5106563.48E−5	5186070.97E−5	5106566.51E−5	5178416.81E−5	5106566.51E−5	5652460.19E−5
	Runtime			6634286.95E−7	**490422037341.26E−13**	**7037475.66E−07**	1600534.22E−5	**2793804.53E−7**	1164833.92E−5
Overall	Objective	$U(\vec{x})$		0/1	**1/1**	0/1	**1/1**	0/1	**1/1**
		$f(\vec{x})$		**1/1**	0/1	**1/1**	0/1	**1/1**	0/1
	Constraints	3/6		**6/6**	3/6	**6/6**	2/6		**6/6**
	Feasibility	0/1		**1/1**	0/1	**1/1**	0/1		**1/1**
	Runtime	0/1		**1/1**	**1/1**	0/1	**1/1**		0/1

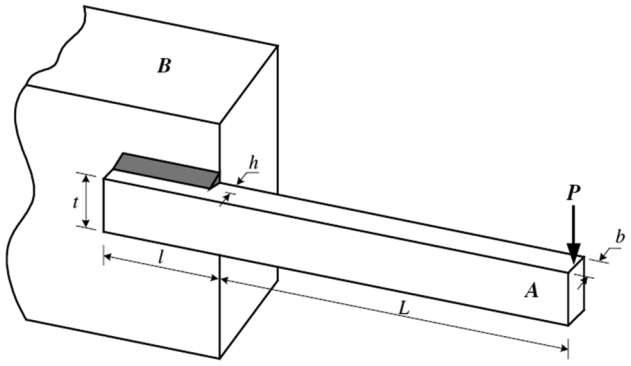


Fig. 20. Welded-beam design engineering optimization problem.

$$\begin{pmatrix}\beta_{1Shroud}\\ \beta_{1Hub}\\ \gamma_{mid}\\ \beta_2\end{pmatrix} \in \begin{bmatrix}30^\circ & 70^\circ\\ 0^\circ & 30^\circ\\ 60^\circ & 89^\circ\\ 40^\circ & 60^\circ\end{bmatrix}$$

There are plenty of ways such as the ones applied in the following studies (Kashan, 2015; Pakzad-Moghaddam, 2016; Shishehgar, Tavakkoli-Moghaddam, Siadat, & Mohammadi, 2014; Tavakkoli-Moghaddam, Shishegar, Siadat, & Mohammadi, 2016) to approach a bi-objective optimization problem. In order to build the Pareto optimal front, here, the following weighting procedure which is adopted from the work of Kashan (2015) is applied.

$$\text{Minimum} - w_i \times \eta + (1 - w_i) \times NPSHr \quad w_i = 0.0001i \qquad i = 0, 1, ...,10000 \tag{17}$$

A standard Multi-Objective Particle Swarm Optimization (MOPSO) with the following parameters was applied in order to solve the problem: $c_1 = 1.5$, $c_2 = 2.5$, swarm size = 100, maximum iterations = 150; Vmin = −4 and Vmax = 4. The above optimization problem was also solved by Boosted MOPSO (BMOPSO), and Pareto optimal fronts obtained by MOPSO and BMOPSO, as well as their combined Pareto fronts, are displayed in Fig. 18.

The number of non-dominated solutions found by MOPSO was 1567, while BMOPSO managed to find 2852 non-dominated solutions. The superiority of the results obtained by BMOPSO, compared to those of MOPSO, can be easily observed in the combined Pareto front. In fact, while MOPSO was only able to find < 55% of the solutions in the combined Pareto front, BMOPSO was able to find > 99% of the solutions.

- **A real-world pressure vessel design engineering problem**

A core pressure vessel design problem concerns minimizing the total costs of the material, while forming and welding the cylindrical vessel (Deb, 2000), as analyzed previously by Sandgren (1990), Wu and Chow (1995), and Lee and Geem (2005). A schematic view of a pressure vessel and its design variables are shown in Fig. 19.

The mathematical formulation of this design problem is as follows:

$$\begin{aligned}
&\min \quad f(\vec{x}) = 0.6224T_s \times R \times L + 1.7781T_h \times R^2 + 3.1611T_s^2 \times L + 19.84T_s^2 \times R\\
&s.\ t.\\
&g_1(\vec{x}) = 0.0193R - T_s \leqslant 0\\
&g_2(\vec{x}) = 0.00954R - T_h \leqslant 0\\
&g_3(\vec{x}) = 750.0 \times 1728.0 - \pi R^2 L - \tfrac{4}{3}\pi R^3 \leqslant 0\\
&g_4(\vec{x}) = L - 240.0 \leqslant 0\\
&g_5(\vec{x}) = 1.1 - T_s \leqslant 0\\
&g_6(\vec{x}) = 0.6 - T_h \leqslant 0\\
&\begin{pmatrix}T_s\\ T_h\\ R\\ L\end{pmatrix} \in \begin{bmatrix}0.0625i\\ 0.0625j\\ 40 & 80\\ 20 & 60\end{bmatrix} \quad where \quad i, j \in \{1, 2, ...\}
\end{aligned} \tag{18}$$

Table 11
Interior point, SQP and active set methods versus their boosted versions in solving welded-beam design optimization problems.

Problem number	Results	Interior point method	Boosted interior point method	SQP	Boosted SQP	Active set		Boosted active set
Welded-beam design	Objective	$U(\vec{x})$	2723933.24E−6	**2723904.79E−6**	2723907.19E−6	**2723904.47E−6**	2919380.37E−6	**2919376.74E−6**
		$f(\vec{x})$	2723933.24E−6	**2723904.79E−6**	**2723904.47E−6**	**2723904.47E−6**	**2919376.74E−6**	**2919376.74E−6**
	Constraints	$g(\vec{x})$	**−3728055.09E−8**	**−4756651.91E−10**	1818989.40E−18	**−3637978.81E−18**	**0.0036379781.81E−18**	**−1818989.40E−18**
			−2094753.23E−6	**−2048983.77E−8**	**−3637978.81E−18**	**−2728484.10E−16**	**0.00**	**−3637978.81E−18**
			−1810970.12E−12	**−2309267.41E−14**	2775557.56E−23	**−2109423.75E−21**	**−1818989.40E−18**	**0.00**
			−1211922.72E−7	**−1203380.14E−9**	9094947.02E−19	**−1909938.87E−17**	**−2337445.79E−7**	**−9094947.02E−19**
			−2337462.59E−7	**−2337445.95E−7**	**−2337445.79E−7**	**−2337445.79E−7**		**−2337445.79E−7**
	Feasibility		**Feasible**	**Feasible**	Infeasible	**Feasible**	Infeasible	**Feasible**
	Decision variables	$\vec{x}$	2450032.03E−6	2450053.31E−6	2450053.57E−7	2450053.57E−7	2450053.57E−7	2450053.57E−7
			8412277.11E−6	8412484.34E−6	8412486.01E−6	8412486.01E−6	9612013.36E−6	9612013.36E−6
			8199397.38E−6	8199113.77E−6	8199110.97E−6	8199110.97E−6	8199110.97E−6	8199110.97E−6
			2450050.14E−6	2450053.54E−6	2450053.57E−7	2450053.57E−7	2450053.57E−7	2450053.57E−7
			2610133.74E−7	**1385521.71E−8**	**1899092.13E−8**	1033230.08E−7	1137788.36E−7	**5822870.55E−8**
	Runtime		0/1	1/1	0/1	**1/1**	0/1	**1/1**
Overall	Objective	$U(\vec{x})$	0/1	1/1	1/1	1/1	**1/1**	1/1
		$f(\vec{x})$	0/1	1/1	1/1	1/1		1/1
	Constraints		**5/5**	2/5	**5/5**	4/5		**5/5**
	Feasibility		**1/1**	0/1	**1/1**	0/1		**1/1**
	Runtime		0/1	**1/1**	0/1	0/1		**1/1**

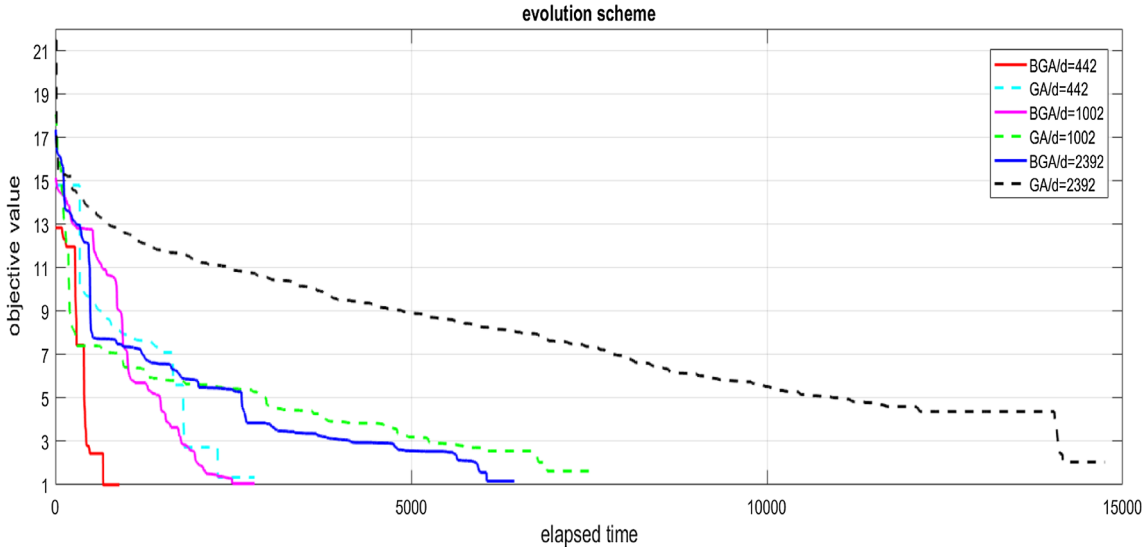


Fig. 21. GA and BGA results for TSPs adopted from TSPLIB.

Table 12
GA versus BGA performance in solving TSPs adopted from TSPLIB.

Solver/solver/problem	Global optimum	Search radius/dimension	GA		BGA	
			Objective value	Runtime (s)	Objective value	Runtime (s)
GA/BGA/TSP	50,778	−/442	68,181	2353	50,778	729
	259,045	−/1002	423,700	7152	273,478	2646
	378,032	−/2392	774,737	14,446	442,155	6212

The results that were obtained are presented in Table 10. It was clear that although none of the traditional methods were able to reach a final feasible solution, the OBA boost mechanism allowed them to find solutions that were both optimum and feasible.

- **A real-world welded-beam design engineering problem**

The performance of many optimization methods have been tested via the welded-beam design problem (Deb, 1991, 2000; Lee & Geem, 2005; Ragsdell & Phillips, 1976; Ravindran, Reklaitis, & Ragsdell, 2006; Siddall, 1972), and its engineering structure is depicted in Fig. 20.

The following mathematical formulation was used to find a feasible set of dimensions h, l, t, and b to carry a certain load (P), while maintaining the minimum total fabrication cost (Deb, 2000).

$$
\begin{aligned}
&f(x) = 1.10471h^2l + 0.04811tb(L+l)\\
&\qquad s.\ t.\\
&g_1(x) = \tau(x) - \tau_d \leqslant 0 \qquad \begin{pmatrix} h \\ l \\ t \\ b \end{pmatrix} \in \begin{bmatrix} 0.125 & 80 \\ 0.1 & \infty \\ 0 & 10 \\ 0.1 & 5 \end{bmatrix}\\
&g_2(x) = \sigma(x) - \sigma_d \leqslant 0\\
&g_3(x) = h - b \leqslant 0\\
&g_4(x) = P - P_c(x) \leqslant 0\\
&g_5(x) = \delta(x) - 0.25 \leqslant 0
\end{aligned}
$$

$$
\text{where} \quad \tau(x) = \sqrt{\frac{(\tau'(x))^2 + (\tau''(x))^2 + l\tau'(x)\tau''(x)}{\sqrt{0.25(l^2+(h+t)^2)}}}, \qquad P = 6000,
$$

$$
\sigma(x) = \frac{504000}{t^2b}, \qquad P_c(x) = 64746(1 - 0.0282346t)tb^3, \qquad \delta(x) = \frac{2.1952}{t^3b}
$$

$$
\tau' = \frac{6000}{\sqrt{2}hl} \qquad \text{and} \qquad \tau'' = \frac{6000(14+0.5l)\sqrt{0.25(l^2+(h+t)^2)}}{2\{0.707hl(\frac{l^2}{12}+0.25(h+t)^2)\}}. \tag{19}
$$

Table 11 shows the significant improvement that was made by the COBA in improving the quality of the solutions found by the Interior Point, SQP and Active Set methods.

As shown in Table 11, applying the COBA helped these methods to improve their f and U optimum values. Furthermore, of particular interest was that boosting these methods resulted in significant decreases in their deviations from feasible solution spaces.

- **Traveling Salesman Problem: a real-world application of discrete optimization**

In order to investigate performance of OBA searching discrete solution space, three Travelling Salesman Problems, with 442, 1002 and 2392 cities that are provided by TSPLIB are solved in this section. Genetic Algorithm (GA) is applied to solve these problems before and after receiving a boost. Chromosomes here are defined as permutations of the number of cities. Next, a proper definition of neighborhood is given to be applied on TSP's discrete search space through OBA. Hereby, neighborhoods are generated by fixing a sub-set of genes to their current values where GA looks for the optimum permutation for the rest of the genes to generate a full chromosome. After discovering the best solution that is placed in a neighborhood through investment optimization step, new neighborhood is established around it by randomly fixing some of its genes to their current values and blanking the rest. Afterwards, in order to calculate return values, forward-looking sample solutions are generated within the same neighborhood by applying crossover and mutation operators on the current best solution in the neighborhood (i.e. neighborhood's center). In addition, test neighborhoods are iteratively updated by crossing over their centers with the center of bull neighborhoods. Applied mutation and cross over functions are elaborated in details as follows.

Let's assume a TSP in which the route to visit 10 cities is supposed to be optimized. Then, a sample neighborhood is as follows.

Sample neighborhood	Fixed genes				Blank spaces					
	6	3	7	8						

Where fixed genes are the ones that GA is not allowed to touch during Investment Optimization step. On the other hand, through applied BGA, Genetic Algorithm is tasked with finding optimum permutation of the remained cities to be placed across the blank spaces which hereafter are referred to as floating genes. For further clarification of the approach, applied mutation and crossover operators are extensively elaborated through following examples.

Sample chromosome	Fixed genes				Floating genes					
	6	3	7	8	5	1	2	4	9	10

In order to mutate a chromosome and ensuring the result not jumping out of the neighborhood, only floating genes are swapped. After performing a predetermined number of swaps on the floating genes the fully mutated chromosome will be ready for evaluation. For example, a two-step mutation on the abovementioned sample chromosome is done through following steps.

Two-step mutation	Fixed genes				Floating genes					
One-time mutated chromosome	6	3	7	8	9	1	2	4	5	10
Fully mutated chromosome	6	3	7	8	9	1	5	4	2	10

Where, swapped genes are highlighted in yellow-color. Again, in order to keep searching within the boundaries of the same neighborhood, crossover operator is only applied to the floating genes. First, floating genes of the parents are divided in half.

Parents	Fixed genes				Floating genes					
					First half			Second half		
Parent #1	6	3	7	8	5	1	2	4	9	10
Parent #2	6	3	7	8	9	1	5	2	10	4

Next step is to swap second halves of the floating genes without touching the fixed ones. For which the result is as follows.

Infeasible offspring	Fixed genes				Floating genes					
					First half			Second half		
Offspring #1	6	3	7	8	5	1	2	2	10	4
Offspring #2	6	3	7	8	9	1	5	4	9	10

Since each chromosome is supposed to be a permutation of one to ten, generated offspring are likely to be infeasible. Therefore, following feasibility step takes place to ensure about feasibility of the final result. Through this feasibility step, offspring are searched for duplicate genes, until there is none left. For instance, duplicate genes are highlighted in yellow-color in the following chromosomes. Then, the corrective action is to swap duplicate genes between two offspring.

Infeasible offspring	Fixed genes				Floating genes					
					First half			Second half		
Offspring #1	6	3	7	8	5	1	2	2	10	4
Offspring #2	6	3	7	8	9	1	5	4	9	10

Eventually, obtained feasible offspring are as follows.

Feasible offspring	Fixed genes				Floating genes					
					First half			Second half		
Offspring #1	6	3	7	8	5	1	9	2	10	4
Offspring #2	6	3	7	8	2	1	5	4	9	10

Obtained results are shown in Fig. 21 and Table 12.

For better understanding, relative objective values $\left(\frac{Best\ Solution}{Global\ Optimum}\right)$ are depicted through Fig. 21. Both of Fig. 21 and Table 12, yield significant superiority of Boosted GA (i.e. BGA) compared to the performance of its traditional form. Despite the NP-hardness of the applied TSPs with up to 2392 cities, BGA has found the global optimum solution in one case and local optima that are very close to the global optimum solution in both two other cases. Where traditional GA stuck in considerably less optimum local optima in all cases. BGA not only is observed to be capable of finding far better solutions but also its runtimes are significantly lower than that of GA. In summary, for almost any amount of elapsed time, BGA was able to deliver far better solutions than the one being delivered by the traditional GA.

5. Conclusion and future work

The boosted optimization algorithm looks for neighborhoods covering optimal solutions. Then, it searches for optimal solutions placed within these neighborhoods. Since the OBA equips optimization methods with human-inspired intelligence, boosted optimization algorithms are definitely expected to perform to a superior level. The performance of the OBA was investigated while boosting fourteen well-known optimization algorithms, including the steepest descent, BTRG, BFGS, Nelder-Mead, PSO, DE, SA, GA, Interior Point, SQP and Active Set methods in optimizing several non-linear continuous/discrete constrained/unconstrained single-objective/multi-objective benchmark problems. Some were found to have smooth surfaces, and others had many local optima; however, the number of variables varied from 2 to 500 in applied continuous problems and the number of cities varied from 442 to 2392 in applied TSPs. As a result of receiving a boost, the performance of each of the tested algorithms was markedly improved,

in terms of feasibility, optimality and/or efficiency. Ultimately, boosted algorithms often provided better solutions, while consuming less time.

Since the OBA has been designed to explore different solution subspaces simultaneously, it lends itself very well to parallel computations; however, further studies are needed to investigate the performance of parallel boosted algorithms. Devising powerful hyper-heuristics by applying OBA on top of the essence of known heuristics, is a very interesting path to future expansions of OBA applications. Since, in theory, there is no limit for boosting exact optimization methods, efforts should also be made, in future, to boost exact optimization methods as well. The results of coupling the quality of exact solutions with the agility of the OBA is expected to be encouraging.

Appendix A. Supplementary material

Supplementary data to this article can be found online at https://doi.org/10.1016/j.cie.2019.07.046.

References

Atashpaz-Gargari, E., & Lucas, C. (2007). Imperialist competitive algorithm: an algorithm for optimization inspired by imperialistic competition. *2007 IEEE congress on evolutionary computation* (pp. 4661–4667). IEEE.

Brusco, M. J. (2015). An iterated local search heuristic for cell formation. *Computers & Industrial Engineering, 90*, 292–304.

Byrd, R. H., Gilbert, J. C., & Nocedal, J. (2000). A trust region method based on interior point techniques for nonlinear programming. *Mathematical programming, 89*(1), 149–185.

Byrd, R. H., Hribar, M. E., & Nocedal, J. (1999). An interior point algorithm for large-scale nonlinear programming. *SIAM Journal on Optimization, 9*(4), 877–900.

Ceschia, S., Di Gaspero, L., & Schaerf, A. (2017). Solving discrete lot-sizing and scheduling by simulated annealing and mixed integer programming. *Computers & Industrial Engineering, 114*, 235–243.

Cheng, M. Y., & Prayogo, D. (2014). Symbiotic organisms search: A new metaheuristic optimization algorithm. *Computers & Structures, 139*, 98–112.

Deb, K. (1991). Optimal design of a welded beam via genetic algorithms. *AIAA Journal, 29*(11), 2013–2015.

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer Methods in Applied Mechanics and Engineering, 186*(2–4), 311–338.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). Positive feedback as a search strategy.

Dorigo, M. (1992). Optimization, learning and natural algorithms. PhD Thesis, Politecnico di Milano, Italy.

Eberhart, R., & Kennedy, J. (1995). Particle swarm optimization. *Proceedings of the IEEE international conference on neural networks* (pp. 1942–1948). .

Fathollahi-Fard, A. M., Hajiaghaei-Keshteli, M., & Tavakkoli-Moghaddam, R. (2018). The social engineering optimizer (SEO). *Engineering Applications of Artificial Intelligence, 72*, 267–293.

Fisher, M. L. (1985). An applications oriented guide to Lagrangian relaxation. *Interfaces, 15*(2), 10–21.

Fletcher, R. (1987). *Practical methods of optimization.* New York: John Wiley & Sons 80.

Focacci, F., Laburthe, F., & Lodi, A. (2003). Local search and constraint programming. *Handbook of metaheuristics* (pp. 369–403). Boston, MA: Springer.

Formato, R. A. (2007). Central force optimization. *Prog Electromagn Res, 77*, 425–491.

Gao, L., Zhang, G., Zhang, L., & Li, X. (2011). An efficient memetic algorithm for solving the job shop scheduling problem. *Computers & Industrial Engineering, 60*(4), 699–705.

Geem, Z. W., Kim, J. H., & Loganathan, G. V. (2001). A new heuristic optimization algorithm: Harmony search. *Simulation, 76*(2), 60–68. https://doi.org/10.1177/003754970107600201.

Gill, P. E., Murray, W., & Wright, M. H. (1981). Practical optimization.

Glover, F. (1989). Tabu search—Part I. *ORSA Journal on Computing, 1*(3), 190–206.

Glover, F. (1990). Tabu search—Part II. *ORSA Journal on Computing, 2*(1), 4–32.

Guo, Q., & Tang, L. (2019). Modelling and discrete differential evolution algorithm for order rescheduling problem in steel industry. *Computers & Industrial Engineering, 130*, 586–596.

He, S., Wu, Q. H., & Saunders, J. R. (2009). Group search optimizer: An optimization algorithm inspired by animal searching behavior. *IEEE Transactions on Evolutionary Computation, 13*(5), 973–990.

Hock, W., & Schittkowski, K. (1983). A comparative performance evaluation of 27 nonlinear programming codes. *Computing, 30*(4), 335–358.

Holland, J. H. (1992). Genetic algorithms. *Scientific American, 267*(1), 66–73.

Jamil, M., & Yang, X. S. (2013). A literature survey of benchmark functions for global optimization problems. Available from: arXiv preprint arXiv:1308.4008.

Jia, Z. H., Zhang, H., Long, W. T., Leung, J. Y., Li, K., & Li, W. (2018). A meta-heuristic for minimizing total weighted flow time on parallel batch machines. *Computers & Industrial Engineering, 125*, 298–308.

Kaasgari, M. A., Imani, D. M., & Mahmoodjanloo, M. (2017). Optimizing a vendor managed inventory (VMI) supply chain for perishable products by considering discount: Two calibrated meta-heuristic algorithms. *Computers & Industrial Engineering, 103*, 227–241.

Kallioras, N. A., Lagaros, N. D., & Avtzis, D. N. (2018). Pity beetle algorithm–A new metaheuristic inspired by the behavior of bark beetles. *Advances in Engineering Software, 121*, 147–166.

Karaboga, D., & Akay, B. (2009). A comparative study of artificial bee colony algorithm. *Applied mathematics and computation, 214*(1), 108–132.

Karlin, S. (1955). The structure of dynamic programing models. *Naval Research Logistics Quarterly, 2*(4), 285–294.

Kashan, A. H. (2015). A new metaheuristic for optimization: Optics inspired optimization (OIO). *Computers & Operations Research, 55*, 99–125.

Kashan, A. H., Tavakkoli-Moghaddam, R., & Gen, M. (2019). Find-Fix-Finish-Exploit-Analyze (F3EA) meta-heuristic algorithm: An effective algorithm with new evolutionary operators for global optimization. *Computers & Industrial Engineering, 128*, 192–218.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *Science, 220*(4598), 671–680.

Kiziloz, H. E., & Dokeroglu, T. (2018). A robust and cooperative parallel tabu search algorithm for the maximum vertex weight clique problem. *Computers & Industrial Engineering, 118*, 54–66.

Laha, D., & Gupta, J. N. (2016). A Hungarian penalty-based construction algorithm to minimize makespan and total flow time in no-wait flow shops. *Computers & Industrial Engineering, 98*, 373–383.

Laha, D., & Gupta, J. N. (2018). An improved cuckoo search algorithm for scheduling jobs on identical parallel machines. *Computers & Industrial Engineering, 126*, 348–360.

Lawler, E. L., & Wood, D. E. (1966). Branch-and-bound methods: A survey. *Operations Research, 14*(4), 699–719.

Lee, K. S., & Geem, Z. W. (2005). A new meta-heuristic algorithm for continuous engineering optimization: Harmony search theory and practice. *Computer Methods in Applied Mechanics and Engineering, 194*(36–38), 3902–3933.

Li, X., Zhang, J., & Yin, M. (2014). Animal migration optimization: An optimization algorithm inspired by animal migration behavior. *Neural Computing and Applications, 24*(7–8), 1867–1877.

Liao, Q., Fan, Q. Q., & Li, J. J. (2019). Translation control of an immersed tunnel element using a multi-objective differential evolution algorithm. *Computers & Industrial Engineering, 130*, 158–165.

Ma, L., Cheng, S., Wang, X., Huang, M., Shen, H., He, X., & Shi, Y. (2017). Cooperative two-engine multi-objective bee foraging algorithm with reinforcement learning. *Knowledge-Based Systems, 133*, 278–293.

Madani-Isfahani, M., Tavakkoli-Moghaddam, R., & Naderi, B. (2014). Multiple cross-docks scheduling using two meta-heuristic algorithms. *Computers & Industrial Engineering, 74*, 129–138.

Marandi, F., & Ghomi, S. F. (2019). Network configuration multi-factory scheduling with batch delivery: A learning-oriented simulated annealing approach. *Computers & Industrial Engineering, 132*, 293–310.

Mirjalili, S. (2015). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems, 89*, 228–249.

Mirjalili, S. (2016). SCA: A sine cosine algorithm for solving optimization problems. *Knowledge-Based Systems, 96*, 120–133.

Mirjalili, S., Jangir, P., Mirjalili, S. Z., Saremi, S., & Trivedi, I. N. (2017). Optimization of problems with multiple objectives using the multi-verse optimization algorithm. *Knowledge-Based Systems, 134*, 50–71.

Mirjalili, S., & Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software, 95*, 51–67.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Caltech concurrent computation program, C3P Report, 826, 1989.

Nourbakhsh, A., Safikhani, H., & Derakhshan, S. (2011). The comparison of multi-objective particle swarm optimization and NSGA II algorithm: Applications in centrifugal pumps. *Engineering Optimization, 43*(10), 1095–1113.

Oftadeh, R., Mahjoob, M. J., & Shariatpanahi, M. (2010). A novel meta-heuristic optimization algorithm inspired by group hunting of animals: Hunting search. *Computers & Mathematics with Applications, 60*(7), 2087–2098.

Pakzad-Moghadam, S. H., Shahmohammadi, M. S., & Ghodsi, R. (2012). A Low-Order Knowledge-Based Algorithm (LOKBA) to solve binary integer programming problems. *International Journal of Academic Research in Business and Social Sciences, 2*(1), 247.

Pakzad-Moghaddam, S. H. (2016). A Lévy flight embedded particle swarm optimization for multi-objective parallel-machine scheduling with learning and adapting considerations. *Computers & Industrial Engineering, 91*, 109–128.

Pakzad-Moghaddam, S. H., Mina, H., Iranmanesh, S. H., & Keyvandarian, A. (2015). A new mathematical model for single machine scheduling with learning effect: Continuous approach. *International Journal of Mathematics in Operational Research, 7*(3), 348–360.

Pakzad-Moghaddam, S. H., Mina, H., & Tavakkoli-Moghaddam, R. (2014). An approach for modeling a new single machine scheduling problem with deteriorating and learning effects. *Computers & Industrial Engineering, 78*, 33–43.

Pan, W. T. (2012). A new fruit fly optimization algorithm: Taking the financial distress model as an example. *Knowledge-Based Systems, 26*, 69–74.

Powell, M. J. (1983). Variable metric methods for constrained optimization. *Mathematical programming the state of the art* (pp. 288–311). Berlin, Heidelberg: Springer.

Puchinger, J., & Raidl, G. R. (2005). Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. *International work-conference on the interplay between natural and artificial computation* (pp. 41–53). Berlin, Heidelberg: Springer.

Ragsdell, K. M., & Phillips, D. T. (1976). Optimal design of a class of welded structures using geometric programming. *Journal of Engineering for Industry, 98*(3), 1021–1025.

Rao, R., & Patel, V. (2013). Comparative performance of an elitist teaching-learning-based optimization algorithm for solving unconstrained optimization problems.

International Journal of Industrial Engineering Computations, 4(1), 29–50.
Rao, R. V., Savsani, V. J., & Vakharia, D. P. (2011). Teaching–learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design, 43*(3), 303–315.
Ravindran, A., Reklaitis, G. V., & Ragsdell, K. M. (2006). *Engineering optimization: Methods and applications.* John Wiley & Sons.
Rechenberg, I. (1978). Evolutionsstrategien. *Simulationsmethoden in der Medizin und Biologie* (pp. 83–114). Berlin, Heidelberg: Springer.
Salimi, H. (2015). Stochastic fractal search: A powerful metaheuristic algorithm. *Knowledge-Based Systems, 75*, 1–18.
Sandgren, E. (1990). Nonlinear integer and discrete programming in mechanical design optimization. *Journal of Mechanical Design, 112*(2), 223–229.
Shareef, H., Ibrahim, A. A., & Mutlag, A. H. (2015). Lightning search algorithm. *Applied Soft Computing, 36*, 315–333.
Shayanfar, H., & Gharehchopogh, F. S. (2018). Farmland fertility: A new metaheuristic algorithm for solving continuous optimization problems. *Applied Soft Computing, 71*, 728–746.
Shishegar, S., Duchesne, S., & Pelletier, G. (2018). Optimization methods applied to stormwater management problems: A review. *Urban Water Journal, 15*(3), 276–286.
Shishehgar, S., Tavakkoli-Moghaddam, R., Siadat, A., & Mohammadi, M. (2014). Design of a dynamic bi-objective relief routing network in the earthquake response phase. *2014 IEEE international conference on industrial engineering and engineering management* (pp. 325–329). IEEE.
Siddall, J. N. (1972). *Analytical decision-making in engineering design.* Prentice Hall.
Storn, R., & Price, K. (1997). Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization, 11*(4), 341–359.
Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y. P., Auger, A., & Tiwari, S. (2005). Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. KanGAL report, 2005005, 2005.
Tamura, K., & Yasuda, K. (2011). Primary study of spiral dynamics inspired optimization. *IEEJ Transactions on Electrical and Electronic Engineering, 6*(S1), S98–S100.
Tavakkoli-Moghaddam, R., Shishegar, S., Siadat, A., & Mohammadi, M. (2016). Design of a reliable bi-objective relief routing network in the earthquake response phase. *Procedia Computer Science, 102*, 74–81.
Waltz, R. A., Morales, J. L., Nocedal, J., & Orban, D. (2006). An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical Programming, 107*(3), 391–408.
Wang, G. G., Gao, X. Z., Zenger, K., & Coelho, L. D. S. (2018). A novel metaheuristic algorithm inspired by rhino herd behavior. *Proceedings of the 9th EUROSIM congress on modelling and simulation, EUROSIM 2016, the 57th SIMS conference on simulation and modelling SIMS 2016* (pp. 1026–1033). Linköping University Electronic Press.
Wolsey, L. A., & Nemhauser, G. L. (2014). *Integer and combinatorial optimization.* John Wiley & Sons.
Wu, S. J., & Chow, P. T. (1995). Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. *Engineering Optimization+ A35, 24*(2), 137–159.
Yang, X. S., & Deb, S. (2009). Cuckoo search via Lévy flights. *2009 World congress on Nature & Biologically Inspired Computing (NaBIC)* (pp. 210–214). IEEE.
Yao, X., Liu, Y., & Lin, G. (1999). Evolutionary programming made faster. *IEEE Transactions on Evolutionary Computation, 3*(2), 82–102.
Yazdani, M., & Jolai, F. (2016). Lion optimization algorithm (LOA): A nature-inspired metaheuristic algorithm. *Journal of Computational Design and Engineering, 3*(1), 24–36.
Zhang, M., Luo, W., & Wang, X. (2008). Differential evolution with dynamic stochastic selection for constrained optimization. *Information Sciences, 178*(15), 3043–3074.