Yokohama Publishers
ISSN 1349-8169 ONLINE JOURNAL

Pacific Journal of Optimization
Vol. 12, No. 1, pp. 151–184, January 2016

HYBRID SIMULATED ANNEALING AND PATTERN SEARCH METHOD FOR SOLVING MINIMAX AND INTEGER PROGRAMMING PROBLEMS

Ahmed F. Ali* and Mohamed A. Tawhid†

Abstract: In this paper, we propose a hybrid simulated annealing with direct search methods (HSAPS) in order to solve integer programming and minimax problems. In HSAPS, we combine the SA with its high ability of global search with the pattern search to refine the simulated annealing solution at each iteration and the Nelder-Mead method in the final stage of the algorithm as a final intensification process in order to avoid running the algorithm with more iterations without any improvement in the search and to accelerate the convergence. Moreover, we investigate the general performance of HSAPS on 17 benchmark functions (7 integer programming problems and 10 minimax problems) and compare HSAPS against other 9 algorithms. Our experimental results demonstrate that HSAPS is a promising algorithm and can solve these type of functions efficiently and in reasonable time.

Key words: *simulated annealing, direct search methods, pattern search method, Nelder-Mead method, integer programming problems, minimax problems.*

Mathematics Subject Classification: *90C47, 90C10, 90C59, 90C56.*

1 Introduction

In the 1970s, metaheuristics methods have been emerged to combine basic heuristic methods in higher level frameworks to explore a search space in an efficient and effective way. Metaheuristics methods have two classes, population based methods and single solution based methods. The population based methods include but not restricted to Ant Colony Optimization (ACO) [9], Genetic Algorithms (GA) [19], Particle Swarm Optimization (PSO) [26], Scatter Search (SS) [15], etc, while the single solution based methods include but not restricted to Tabu Search (TS) [14], Simulated Annealing (SA) [25], Variable Neighborhood Search (VNS) [32], [33], Iterated Local Search (ILS) [45]. The main key feature of designing any methaheuristics algorithm is its capability of performing wide exploration and deep exploitation. There are several different single solution based methods, also called trajectory methods, which can be seen as an extension of local search algorithms. The goal of this kind of metaheuristic is to escape from local minima in order to proceed in the exploration of the search space and to move on to find better local minima such as TS, ILS, VNS and

*The postdoctoral fellowship of the 1st author is supported by NSERC.
†The research of the 2nd author is supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC).

SA. Metaheuristics have been applied to solve many problems in different fields such as engineering, economics, management, biology and combinatorial optimization problems, see, e.g., [35], [41]. SA is one of the most applied metaheuristics methods because of its robustness and high ability to escape from trapping in local minima, however SA suffers from slow convergence because it explores the search space randomly without saving information about promising search direction [35]. In order to overcome the slow conference problem of SA, many researchers have tried to integrate SA with other local search methods such as Nelder-Mead method and pattern search method to increase its convergence, see, e.g., [6], [7], [16], [17].

The above mentioned algorithms have been widely used to solve unconstrained and constrained problems and their applications. However much less work has been done on algorithms for integer programming and minimax problems. There are many real life applications such as Warehouse location problem, VLSI (very large scale integration) circuits design problems, robot path planning problems, scheduling problem, game theory, engineering design problems can be formulated as integer programming and minimax problems, see, e.g., [8], [36], [51].

Branch and Bound (BB) is the most popular methods for solving integer programming problems, however BB suffers from high computational cost, since it explores a search tree containing hundreds or more nodes on large scale problems.

Recently there have been significant research efforts to apply some of swarm intelligence algorithms such as ant colony algorithm [23], [24], artificial bee colony algorithm [1], [47], particle swarm optimization algorithm [38], cuckoo search algorithm [48] and firefly algorithm[2] to solve integer programming problems.

There are some algorithms based on a smooth techniques have been applied for solving minimax problems. These techniques are solving a sequence of smooth problems, which approximate the minimax problems in the limit [29], [39], [50]. The algorithms based in theses techniques aim to generate a sequence of approximations, which converges to Kuhn-Tucker point of the minimax problem, for a decreasing sequence of positive smoothing parameters. However, the drawback of theses algorithms is these parameters are small too fast and the smooth problems become significantly ill-conditioned. Some swarm intelligence algorithms such as PSO [38] have been applied to solve minimax problems. The main drawback of applying swarm intelligence algorithms for solving minimax and integer programming problems is that they are a population based methods which are computationally expensive.

Simulated annealing algorithm (SA) is a popular approximation algorithm that has been applied to a variety of optimization problems. SA has received significant attention in the last two decades to solve optimization problems due to its simplicity and powerful ability to avoid trapping in local minima by accepting the worst solution with a specific probability and making uphill moves, where a desired global minimum/maximum is hidden among many poorer local minima/maxima. These days SA has become one of the many heuristic approaches designed to give a good, not necessarily optimal solution. SA generates a trail solutions with a random moves, however it suffers from slow convergence.

Pattern search (PS) is a family of numerical optimization methods that do not require the gradient of the problem to be optimized. Hence PS can be used on functions that are not continuous or differentiable. Such optimization methods are also known as direct-search, derivative-free, or black-box methods. The name, pattern search, was coined by Hooke and Jeeves [20].

In order to overcome the slow convergence of the simulated annealing algorithm, the main objective of this paper is to produce a new hybrid algorithm by combining the simulated annealing algorithm with direct search methods in order to solve integer programming and

minimax problems. In the proposed algorithm, we try to overcome the main problem of the simulated annealing algorithm which is the slow convergence by invoking the pattern search and the Nelder-Mead methods in order to accelerate the search and avoid running the algorithm more iterations around the optimal solution without any improvements.

In this paper, we present a hybrid simulated annealing with direct search method to solve integer programming and minimax problems. The proposed algorithm is called Hybrid Simulated Annealing and Pattern Search (HSAPS) algorithm. HSAPS starts with an initial solution generated randomly and at each temperature, the SA starts to generate a trail solutions, the best trail solution is accepted if its objective function is better than the previous best solutions, otherwise it accepted if its probability is less than or equal the random generated number. The best accepted solution will pass to the pattern search in order to intensify the search around the best solution and as soon as the temperature reaches to the minimum value, the overall best solution will pass to the Nelder-Mead algorithm in order to accelerate the search and accelerate the convergence.

The rest of the paper is organized as follows. In Section 2, we describe the integer programming and the minimax problems. In Section 3, we give an overview of simulated annealing algorithm, pattern search and the Nelder-Mead methods as direct search methods. In Section 4, we present the proposed algorithm. In Section 5, we give the numerical results of the proposed algorithm. In Section 6, we give summary and conclusion.

2 Definition of the Problems and an Overview of the Applied Algorithms

In this section, we highlight the definitions of integer programming and the minimax problems and present the main steps of the simulated annealing algorithm and two of the most wide used direct search methods, pattern search and the Nelder-Mead methods as follows.

2.1 The integer programming problem

An integer programming problem is a mathematical optimization problem in which all of the variables are restricted to be integers. The unconstrained integer programming problem can be defined as follows.

$$minf(x), \quad x \in S \subseteq \mathbb{Z}^n, \tag{2.1}$$

Where $\mathbb{Z}$ is the set of integer variables, S is a not necessarily bounded set.

2.2 Minimax problem

The general form of the minimax problem [50] is defined by:

$$\min \ F(x) \tag{2.2}$$

where

$$F(x) = \max \ f_i(x), \quad i = 1, \ldots, m \tag{2.3}$$

with $f_i(x) : S \subset \mathbb{R}^n \to \mathbb{R}, i = 1, \ldots, m$.
The nonlinear programming problems of the form:

$$\begin{aligned} &\min \ F(x), \\ &g_i(x) \geq 0, \qquad i = 2, \ldots, m, \end{aligned}$$

Table 1: The parameters of the pattern search algorithm.

parameter	definition
Δ_0	Initial mesh size
d	Variable dimension
σ	Reduction factor of mesh size
m	Pattern search repetition number
ε	Tolerance

can be transformed to minimax problems as follows:

$$\min \ \max \ f_i(x), \quad i = 1, \ldots, m \tag{2.4}$$

where

$$\begin{aligned} f_1(x) = &\quad F(x), \\ f_i(x) = &\quad F(x) - \alpha_i g_i(x), \\ \alpha_i > 0, &\quad ; \quad i = 2, \ldots, m \end{aligned} \tag{2.5}$$

It has been proven that for sufficiently large α_i, the optimum point of the minimax problem, coincides with the optimum point of the nonlinear programming problem [3].

2.3 Pattern search method

Direct search method is a method for solving optimization problem that dose not require any information about the gradient of the objective function. Pattern search method is one of the most applied direct search method to solve a global optimization problems. The pattern search method (PS) has been proposed by Hook and Jeeves (HJ) [20]. In PS method, there are two type of moves, the exploratory moves and the pattern moves. In the exploratory moves a coordinate search is applied around a selected solution with a step length of Δ as in Algorithm 1. The exploratory move is considered successful, if the function value of the new solution is better than the current solution, Otherwise the step length is reduced as shown below in Equation 2.6. If the exploratory move is successful, then the pattern search is applied in order to generate the iterate solution. The exploratory move is applied on the iterate solution, if the iterate solution is better than the current solution, then the iterate solution is accepted as a new solution. Otherwise if the exploratory move is unsuccessful, the pattern move is rejected and the step length Δ is reduced. The operation is repeated until termination criteria are satisfied. The algorithm of HJ pattern search and the main steps of it are presented in Algorithm 2 as follows. The parameters of Algorithms 1, 2 are given in Table 1

Let us summarize the pattern search algorithm in the following steps:

Step 1. The algorithm starts by setting the initial values of the mesh size Δ_0, reduction factor of mesh size σ and termination parameter ε.

Step 2. Apply exploratory search as shown in algorithm 1 by calculating $f(x^k)$ in order to obtain a new base point

Step 3. If the exploratory move is successful, perform pattern search move; otherwise check the value of the mesh size Δ, if $\Delta < \varepsilon$, where ε is a very small value, stop the search and produce the current solution.

Algorithm 1 Exploratory search

INPUT: Get the values of x^0, k, Δ_0, d
OUTPUT: New base point x^k

Set $i = 1$
Set $k = 1$
repeat
  Set $x_i^k = x_i^{k-1} + \Delta_{k-1} x_i^{k-1}$
  if $f(x_i^k) < f(x_i^{k-1})$ **then**
    Set $x_i^{k+1} = x_i^k$
  end if
  Set $i = i + 1$
  Set $k = k + 1$
until $i \leq d$

Algorithm 2 The basic pattern search algorithm

INPUT: Get the values of x
OUTPUT: Best solution x^*

Set the values of the initial values of the mesh size Δ_0, reduction factor of mesh size σ and termination parameter ε
Set $k = 1$ {**Parameter setting**}
Set the starting base point x^{k-1} {**Initial solution**}
repeat
  Perform exploratory search as shown in Algorithm 1
  if exploratory move success **then**
    Go to 16
  else
    if $\|\Delta_k\| < \varepsilon$, **then**
      Stop the search and the current point is x^*
    else
      Set $\Delta_k = \sigma \Delta_{k-1}$ {**Incremental change reduction**}
      Go to 5
    end if
  end if
  Perform pattern move, where $x_p^{k+1} = x^k + (x^k - x^{k-1})$
  Perform exploratory move with x_p as the base point
  Set x^{k+1} equal to the output result exploratory move
  if $f(x_p^{k+1}) < f(x^k)$ **then**
    Set $x^{k-1} = x^k$
    Set $x^k = x^{k+1}$ {**New base point**}
    Go to 16
  else
    Go to 9 {**The pattern move is failure**}
  end if
  Set $k = k + 1$
until $k \leq m$

Step 4. If the exploratory move is unsuccessful and Δ is not less than ε, reduce the mesh size as shown in the following equation

$$\Delta_k = \sigma \Delta_{k-1} \tag{2.6}$$

Step 5. Apply pattern move by calculating x_p, where $x_p^{k+1} = x^k + (x^k - x^{k-1})$.

Step 6. Set x_p as a new base point and apply exploratory move on it.

Step 7. If the pattern move is successful, repeat the pattern search move on the new point; otherwise the pattern search is unsuccessful and reduce the mesh size as shown in Equation 2.6.

Step 8. The steps are repeated until the termination criteria are satisfied (number of iterations).

2.4 Nelder Mead method

The Nelder-Mead algorithm (NM) is proposed by Nelder and Mead in 1965 [34]. NM algorithm is one of the most popular derivative-free nonlinear optimization algorithms. It starts with $n+1$ points (vertices) $x_1, x_2, \ldots, x_{n+1}$. The vertices are evaluated, ordered and re-labeled in order to assign the best point and the worst point. In minimization problems, the x_1 is considered as the best vertex or point if it has the minimum value of the objective function, while the worst point x_{n+1} with the maximum value of the objective function. At each iteration, new points are computed, along with their function values, to form a new simplex. Four scalar parameters must be specified to define a complete Nelder-Mead algorithm: coefficients of reflection ρ, expansion χ, contraction τ, and shrinkage ϕ. These parameters are chosen to satisfy $\rho > 0$, $\chi > 1$, $0 < \tau < 1$, and $0 < \phi < 1$. The main steps of the Nelder-Mead algorithm are presented as shown below in Algorithm 3. The Nelder-Mead algorithm starts with $n+1$ vertices x_i, $i = 1, \ldots, n+1$, which are evaluated by calculation their fitness function values. The vertices are ordered according to their fitness function. The reflection process starts by computing the reflection point $x_r = \bar{x} + \rho(\bar{x} - x_{(n+1)})$, where $\bar{x}$ is the average of all points except the worst. If the reflected point x_r is lower than the n^{th} point $f(x_n)$ and greater than the best point $f(x_1)$, then the reflected point is accepted and the iteration is terminated. If the reflected point is better than the best point, then the algorithm starts the expansion process by calculating the expanded point $x_e = \bar{x} + \chi(x_r - \bar{x})$. If x_e is better than the reflected point n^{th}, the expanded point is accepted; otherwise the reflected point is accepted and the iteration will be terminated. If the reflected point x_r is greater than the n^{th} point x_n the algorithm starts a contraction process by applying an outside x_{oc} or inside contraction x_{ic} depending on the comparison between the values of the reflected point x_r and the n^{th} point x_n. If the contraction point x_{oc} or x_{ic} is greater than the reflected point x_r, the shrink process is starting. In the shrink process, the points are evaluated and the new vertices of simplex at the next iteration will be $x'_2, \ldots, x'_{n+1}$, where $x' = x_1 + \phi(x_i - x_1), i = 2, \ldots, n+1$.

In Figure 1, we present an example in order to explain the main steps of the Nelder-Mead algorithm in two dimensions.

Step 1. Given the current solution x, two neighborhood trial points y_1 and y_2 are generated in a neighborhood of x as in Figure 1 (a).

Step 2. A simplex is constructed in order to find a local trial point as in Figure 1 (b).

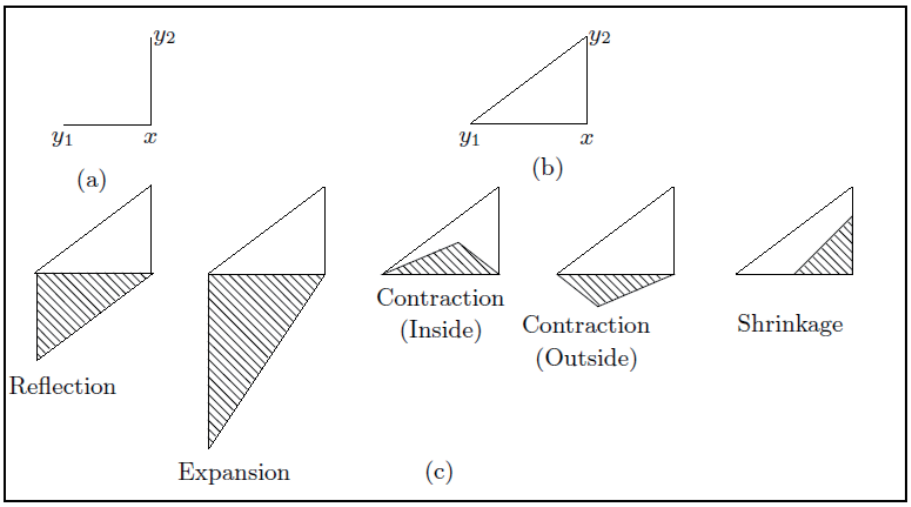

Figure 1: Nelder-Mead search strategy in two dimensions.

Step 3. If y_2 is a worst point, we apply the Nelder-Mead algorithm to find a better movement, as in Figure 1 (c). If we find a better movement, we refer to this point as a local trial point.

3 Simulated Annealing Algorithm

In the following section, we describe the main concepts and structure of the standard simulated annealing algorithm.

3.1 Main concepts

Simulated annealing (SA) is probably the most widely used meta-heuristic algorithms in combinatorial optimization problem. SA was proposed by Kirkpatrick et al. [25]. It was motivated by the analogy between the physical annealing of metals and the process of searching for the optimal solution in a combinatorial optimization problem. It is inspired by the Metropolis algorithm [11]. The main objective of SA method is to escape from local optima and so to delay the convergence.

Many trial solutions are generated at a particular level of temperature in the epoch length (SA inner loop). The operation is repeated until the temperature reaches to its minimum value.

3.2 Cooling schedule

We should deal carefully with the tuning parameters of the cooling schedule in order to improve the performance of SA.

- **Choice of an initial temperature**. Choosing too high temperature will cost computation time expensively, while too low temperature will exclude the possibility of

Algorithm 3 The Nelder-Mead Algorithm

1. Let x_i denote the list of vertices in the current simplex, $i = 1, \ldots, n+1$.
2. Order. Order and re-label the $n+1$ vertices from lowest function value $f(x_1)$ to highest function value $f(x_{n+1})$ so that $f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{n+1})$.
3. Reflection. Compute the reflection point x_r by
$x_r = \bar{x} + \rho(\bar{x} - x_{(n+1)})$, where $\bar{x}$ is the centroid of the n best points,
$\bar{x} = \sum(x_i/n), i = 1, \ldots, n$.
if $f(x_1) \leq f(x_r) < f(x_n)$ **then**
replace x_{n+1} with the reflected point x_r and go to Step 7.
end if
4. Expansion.
if $f(x_r) < f(x_1)$ **then**
Compute the expansion point x_e by $x_e = \bar{x} + \chi(x_r - \bar{x})$.
end if
if $f(x_e) < f(x_r)$ **then**
Replace x_{n+1} with x_e and go to Step 7.
else
Replace x_{n+1} with x_r and go to Step 7.
end if
5. Contraction.
if $f(x_r) \geq f(x_n)$ **then**
Perform a contraction between $\bar{x}$ and the best among x_{n+1} and x_r.
end if
if $f(x_n) \leq f(x_r) < f(x_{n+1})$ **then**
Calculate $x_{oc} = \bar{x} + \tau(x_r - \bar{x})$ {*Outside contract*}
end if
if $f(x_{oc}) \leq f(x_r)$ **then**
Replace x_{n+1} with x_{oc} and go to Step 7.
else
Go to Step 6.
end if
if $f(x_r) \geq f(x_{(n+1)}$ **then**
Calculate $x_{ic} = \bar{x} + \tau(x_{n+1} - \bar{x})$. {*Inside contract*}
end if
if $f(x_{ic}) \geq f(x_{(n+1)}$ **then**
replace x_{n+1} with x_{ic} and go to Step 7.
else
go to Step 6.
end if
6. Shrink. Evaluate the n new vertices
$x' = x_1 + \phi(x_i - x_1), i = 2, \ldots, n+1$.
Replace the vertices $x_2, \ldots, x_{n+1}$ with the new vertices $x'_2, \ldots, x'_{n+1}$.
7. Stopping Condition. Order and re-label the vertices of the new simplex as $x_1, x_2, \ldots, x_{n+1}$ such that $f(x_1) \leq f(x_2) \leq \ldots \leq f(x_{n+1})$
if $f(x_{n+1}) - f(x_1) < \epsilon$ **then**
Stop, where $\epsilon > 0$ is a small predetermined tolerance.
else
Go to Step 3.
end if

ascent steps, thus losing the global optimization feature of the method. We have to balance between these two extreme procedures.

- **Choice of the temperature reduction strategy**. If the temperature is decreased slowly, better solutions are obtained but with a more significant computation time. On the other side, a fast decrement rule causes increasing the probability of being trapped in a local minimum.
- **Equilibrium State**. In order to reach an equilibrium state at each temperature, a number of sufficient transitions (moves) must be applied. The number of iterations must be set according to the size of the problem instance and particularly proportional to the neighborhood size.

3.3 Stopping criteria

Concerning the stopping condition, theory suggests the final temperature is equal to 0. In practice, one can stop the search when the probability of accepting move is negligible, or reaching to the final temperature T_{min}.

3.4 Simulated annealing algorithm

The basic SA algorithm is described below in Algorithm 4. SA proceeds in several iterations from an initial solution x^0, which is generated randomly across the search space given by each search function. At each iteration (SA external loop), a random neighbor solution $x\prime$ is generated at the current temperature (SA inner loop). The neighbor solution that improves the objective function is always accepted. Otherwise, the neighbor solution is selected with a given probability that depends on the current temperature T and the amount of degradation ΔE of the objective function. ΔE represents the difference in the objective value between the current solution x and the generated neighboring solution $x\prime$. This probability follows, in general, by the Boltzmann distribution as in (3.1).

$$P(\Delta E, T) = \exp(\frac{-f(x\prime) - f(x)}{T}). \tag{3.1}$$

Algorithm 4 The basic simulated annealing algorithm

$x = x_0$ {**initial solution**}
$T = T_{max}$ {**starting temperature**}
repeat
  repeat
    Generate a random neighbor $x\prime$ {**at a fixed temperature**}
    $\Delta E = f(x\prime) - f(x)$
    if $\Delta E \leq 0$ **then**
      $x = x\prime$. {**accept the neighbor solution**}
    else
      Accept $x\prime$ with probability $e^{\frac{-\Delta E}{T}}$
    end if
  until (Equilibrium condition) {**e.g. number of iterations executed at each T**}
  $T = g(T)$ {**temperature update**}
until (stopping criteria satisfied) {e.g. $T < T_{min}$}

4 The Proposed HSAPS Algorithm

In this section, we present in Algorithm 5 the main structure of the proposed HSAPS algorithm. Now, we give a brief description of the main components of the proposed HSAPS.

- **Initial temperature**. The starting temperature $T = T_{max}$ is one of the important control parameters in HSAPS algorithm, that has direct effect to accept all neighbors during the initial phase of the method. It must not be too high to conduct a random search for a period of time but high enough to allow moves to almost neighborhood state.
- **Equilibrium state**. In order to reach an equilibrium state at each temperature, a number of moves must be applied, which is determined before the search starts. The equilibrium state of HSAPS is applied in the epoch length (SA inner loop), which based on the number of the generated trail solutions m_1. The high number of generated trail solutions we get, the more expensive cost function we have.
- **Temperature reduction value**.There is a direct relation between the quality of the generated trail solutions and the speed of the cooling schedule. If the temperature is decreased slowly, better solutions are obtained, but high computation time is obtained. The temperature T can be updated, i.e., $T = \lambda T$, where λ is a cooling ratio, $\lambda \in [0.9, 0.99]$.
- **SA stopping condition**. The basic SA method uses one of the following stopping criteria.

  1. The probability of accepting move is negligible.
  2. Achieving a predetermined number of iterations without improvement of the best found solution [43].
  3. Achieving a predetermined number of times a percentage of neighbors at each temperature is accepted.
  4. Reaching the final temperature $T \leq T_{min}$.

  The proposed HSAPS uses different kinds of termination criteria, condition (4) is one of them.
- **Neighborhood radius update**. HSAPS algorithm uses a special parameter called the neighborhood radius parameter z, which has a direct effect to update the solutions. The main role of z in HSAPS is to help the algorithm to achieve the global search and the local search processes. HSAPS starts with initial value $z = z_0$, when HSAPS could obtain an improvement of the trail solutions, the neighborhood area should be expanded by increasing the value of z using a factor $\mu > 1$, the increased value of z is defined in (4.1).

$$z_+ = \mu z. \tag{4.1}$$

  The expanded neighborhood area is controlled by the value of z_+, which is the upper limit by a pre-specified value z_{max}.

  When HSAPS could not obtain any improvement of the trail solutions, the neighborhood area should be reduced by decreasing the value of z, using a factor $\nu < 1$, the decreased value of z is defined in (4.2).

$$z_{-} = \nu z. \tag{4.2}$$

The reduced neighborhood area is controlled by the value of z_{-}, which is the lower limit by a pre-specified value z_{min}.

- **Pattern search parameters**. HSAPS uses PS as a local search method in order to refine the obtained solution from the simulated annealing algorithm. In PS the mesh size is initialized as Δ_0 and when no improvement archived in the exploration search process, the mesh size is deducted by using reduction factor σ. The PS steps are repeated m_2 times in order to increase the exploitation process of the algorithm.
- **Final intensification**. The best obtained solutions from the simulated annealing and the pattern search are collected in list in order to apply the Nelder-Mead method on them, the number of the solutions in these list is called N_{elite}
- **Stopping condition parameters**. The main termination criterion in HSAPS is the completeness of the cooling schedule, however HSAPS maybe terminated if the difference between the function values of the current solution $f(x)$ and the global optimal solution $f(x^*)$ becomes less than $Tol = 10^{-4}$, or the number of maximum iterations exceeds the desired function evaluation number.

We can summarize the main steps of the HSAPS algorithm as follows.

Step 1. The algorithm starts to set the parameter of the simulated annealing (T_{max}, epoch length M, cooling reduction ratio λ and minimum temperature T_{min}), pattern search algorithm parameters (mesh size Δ, reduction factor σ, number of PS iterations m_2).

Step 2. An initial solution x^0 is generated randomly in the range $[L, U]$ for each test function.

Step 3. At a fixed temperature, the SA starts to generate the trail solutions m_1 times. The best trail solution is accepted if its function values is better than the current best solution and the neighborhood radius is expanded and updated using Equation 4.1, otherwise the solution is selected if its probability $P \geq r$, $r \in [0, 1]$ and the neighborhood radius is shrinkage and updated using Equation 4.2.

Step 4. The PS represents the intensification part of the algorithm and it is iterated m_2 times in order to refine the obtained solution from SA by applying Algorithm 2.

Step 5. If the generated pattern search best trail solution is better than the SA best trail solution, we accept the pattern search trail solution, otherwise the algorithm accept the SA trail solution.

Step 6. At the end of the search, when the temperature reached to the minimum temperature $T \leq T_{min}$, the best solution is refined by applying a the Nelder-Mead method in order to refine the overall best obtained solution.

5 Numerical Experiments

The general performance of HSAPS algorithm is presented in order to investigate its efficiency, by comparing the results of HSAPS algorithm against other algorithms. HSAPS was

Algorithm 5 The proposed HSAPS algorithm

Set the values of the initial values of mesh size $\Delta_0 > 0$, reduction factor of mesh size σ, trail points radius z, initial temperature T_{max} epoch length m_1, PS repetition numbers m_2, cooling reduction ratio λ and minimum temperature T_{min}.
Set the initial temperature $T := T_{max}$
Set $k = 0$
Generate the initial solution x^k randomly across the search space of the given test function.
repeat
  repeat
    Set $y = x^k + (2r_1 - 1)z, \quad r_1 \in [0, 1]$ {**Exploration process**}
    Set $\Delta E = f(y) - f(x^k)$
    if $\Delta E < 0$ **then**
      $x_s = y$
      Update the neighborhood radius z using Equation(4.1) {**Expand the search area**}
    else
      Generate $r_2, r_2 \in [0, 1]$
      if $r_2 \leq e^{\frac{-\Delta E}{T}}$ **then**
        $x_s = y$ { Accept the solution y with a probability $e^{\frac{-\Delta E}{T}}$ }
        Update the neighborhood radius d using Equation(4.2). {**shrink the search area**}
      end if
    end if
    Set x_s as the initial solution for pattern search algorithm
    for $(j = 1; j < m_2; j{+}{+})$ **do**
      Apply the pattern search algorithm on the simulated annealing solution x_s as shown in Algorithm 2
    end for{ **Apply pattern search as a local search algorithm**}
    Set x_p equal to the best trail solution from pattern search algorithm.
    if $x_p < x_s$ **then**
      $x^{k+1} = x_p$ {**Accept the pattern search solution as a new solution**}
    else
      $x^{k+1} = x_s$ {**decline the pattern search solution** }
    end if
    $k = k + 1$
  until $(k \leq m_1)$
  $T = \lambda T$
until $T \leq T_{min}$
Apply the Nelder-Mead method as shown in Algorithm 3 on the best fond solution so far {**Final Initialization process**}

programmed in MATLAB, the results of the other comparative algorithms are taken from their original papers. In the following subsections, the parameter setting of HSAPS algorithm with more details and the properties of the applied test functions have been reported. Also the performance analysis of the proposed algorithm is presented in details.

5.1 Parameter setting

Before discussing the experimental results, the parameters setting of HSAPS algorithm are reported in Table 2 and they can be summarized as follows. We summarize the parameters

Table 2: Parameter setting.

Parameters	Definition	Values
T_{max}	Initial temperature	0.9
T_{min}	Minimum temperature	$min(0.01, 0.01T_{max})$
λ	Cooling ratio	0.9
z_0	Initial radius for generating trail solutions	$(z_{max}+z_{min})/2$
z_{min}	Minimum setting of z	$(U-L)/50$
z_{max}	Maximum setting of z	$(U-L)/2$
μ	Radius expansion constant	1.6
ν	Radius shrinkage constant	0.65
Δ_0	Initial mesh size	$(U_i - L_i)/3$
σ	Reduction factor of mesh size	0.01
m_1	No of SA generated trail solution	2
m_2	No of PS local search iterations	d
Tol	Termination tolerance	10^{-4}
N_{elite}	No of best solutions for final intensification	1

setting of HSAPS approach in the following groups.

- **Initial solution**
  HSAPS starts with initial solution x^0 generated randomly across the search space given by each test function, where
  $[L, U] = \{x \in \mathbf{R}^n : l_i \leq x_i \leq u_i, \;\; i = 1, \ldots, n\}$
- **Cooling Schedule**
  The initial temperature T is set high enough to accept all neighbors during the initial phase of the algorithm. We set the initial temperature value $T_{max} = 0.9$ and the cooling ratio λ equal to 0.9. The SA outer loop is terminated when the temperature reaches to its minimum value T_{min}, which is setting to $T_{min} = \min(0.01, 0.01T_{max})$.
- **Trail solution number**
  The number of generated trail solutions depends on the value of m_1 parameter. Increasing the number of m_1 means increasing the number of trail solutions and the value of the evaluation function. The experimental tests show that the best value of the generated trail solutions is equal to $m_1 = 2$.
- **Neighborhood radius parameters**
  The main role of the parameter z is to control the exploration and exploitation processes in the SA algorithm. In order to control the expansion and the shrinkage of the neighborhood zone, the used parameters μ and ν in Equations 4.1 and 4.2 are set

Table 3: The properties of the Integer programming test functions.

Function	Dimension (d)	Bound	Optimal
FI_1	5	[-100 100]	0
FI_2	5	[-100 100]	0
FI_3	5	[-100 100]	-737
FI_4	2	[-100 100]	0
FI_5	4	[-100 100]	0
FI_6	2	[-100 100]	-6
FI_7	2	[-100 100]	-3833.12

equal to 1.6 and 0.65, respectively and the maximum value of the radius z_{max} is set to $(U - L)/2$, while the minimum value of the radius z_{min} is set to $(U - L)/50$.

- **PS parameters**
  The initial mesh size is set to $(U-L)/3$, and when the exploratory move is unsuccessful, the mesh size is reduced by using reduction factor σ, which is set to 0.01. The iteration number of the pattern search is equal to d.

- **Intensification parameters**
  In the final stage of the algorithm, the Nelder-Mead method is applied in order to refine the best found solution so far. We set the number of the best found solutions list N_{elite} to 1.

- **Stopping condition parameters**
  Reaching a final temperature is the main stopping criterion in the proposed HSAPS algorithm, where $T \leq T_{min}$, but there are 2 other termination criteria which HSAPS is applied. The first one is the maximum number of the obtained function evaluation is equal to 20,000, while the second is the difference between the function values of the current solution $f(x)$ and the global optimal value at the global optimal solution $f(x^*)$ is less than $Tol = 10^{-4}$.

5.2 Integer programming problems

HSAPS is tested on 7 benchmark integer programming problems ($FI_1 - FI_7$) in order to verify its efficiency with integer programming problems. The properties of the benchmark functions (function number, dimension of the problem, problem bound and the global optimal of each problem) are listed in Table 3. The problems that were used are:

Test problem 1 ([42]). This problem is defined by

$$FI_1(x) = \|x\|_1 = |x_1| + \ldots + |x_n|$$

Test problem 2 ([42]). This problem is defined by

$$FI_2 = x^T x = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Test problem 3 ([13]). This problem is defined by

$$FI_3 = \begin{bmatrix}15 & 27 & 36 & 18 & 12\end{bmatrix} x + x^T \begin{bmatrix} 35 & -20 & -10 & 32 & -10 \\ -20 & 40 & -6 & -31 & 32 \\ -10 & -6 & 11 & -6 & -10 \\ 32 & -31 & -6 & 38 & -20 \\ -10 & 32 & -10 & -20 & 31 \end{bmatrix} x,$$

Test problem 4 ([13]). This problem is defined by

$$FI_4(x) = (9x_1^2 + 2x_2^2 - 11)^2 + (3x_1 + 4x_2^2 - 7)^2$$

Test problem 5 ([13]). This problem is defined by

$$\begin{aligned} FI_5(x) = \; & (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 \\ & +10(x_1 - x_4)^4 \end{aligned}$$

Test problem 6 ([40]). This problem is defined by

$$FI_6(x) = 2x_1^2 + 3x_2^2 + 4x_1x_2 - 6x_1 - 3x_2$$

Test problem 7 ([13]). This problem is defined by

$$\begin{aligned} FI_7(x) = \; & -3803.84 - 138.08x_1 - 232.92x_2 + 123.08x_1^2 \\ & +203.64x_2^2 + 182.25x_1x_2 \end{aligned}$$

5.3 The efficiency of the proposed HSAPS algorithm with integer programming problems

The first experimental test is applied to verify the power of the proposed HSAPS with integer programming problems. We compare the standard simulated annealing algorithm with the proposed HSAPS algorithm without applying the final intensification process (Nelder-Mead method). We set the same parameter values for both algorithms in order to make a fair comparison. The functions FI_3, FI_5 and FI_6 have been selected (picked randomly) to show the efficiency of the proposed algorithm by plotting the values of function values versus the number of iterations as shown in Figure 2. In Figure 2, the solid line refers to the proposed HSAPS results, while the dotted line refers to the standard simulated annealing results after 50 iterations. The graphs in Figure 2 show that the function values rapidly decrease as the number of iterations increase for HSAPS results than those of the standard simulated annealing algorithm. We can conclude from the graphs in Figure 2 that the combination between the standard simulated annealing algorithm with pattern search method can improve the performance of the standard simulated annealing algorithm and accelerate its convergence.

5.4 The general performance of the HSAPS algorithm with integer programming problems

The second experimental test is applied to investigate the general performance of the proposed algorithm with the integer programming problems by plotting the values of function values versus the number of iterations as shown in Figure 3 for four test functions FI_1, FI_2,

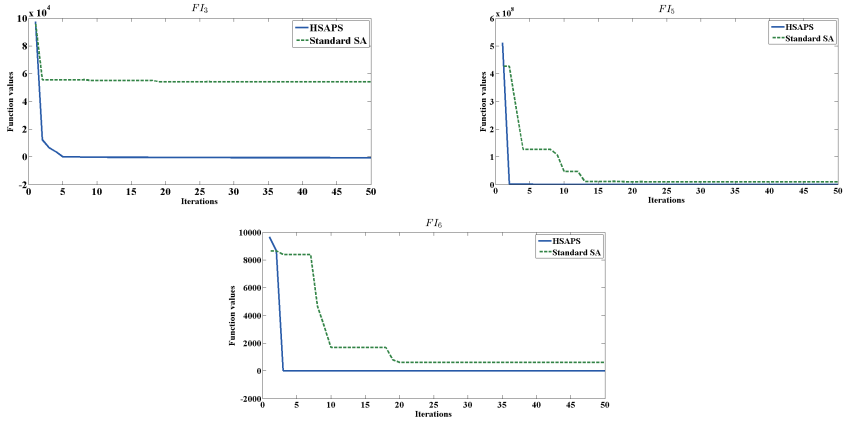

Figure 2: The efficiency of the proposed HSAPS algorithm with integer programming problems

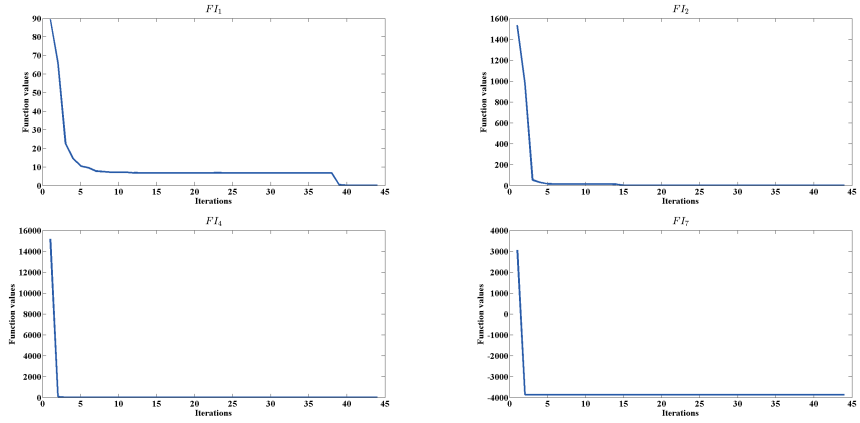

Figure 3: The general performance of HSAPS algorithm with integer programming problems

FI_4 and FI_7. The results in Figure 3 are the results of the proposed algorithm without applying the Nelder-Mead method in the final stage of the algorithm. The graphs in Figure 3 show that the function values of the proposed DSFFA rapidly decrease as the number of iterations increase. We note that the hybridization between the simulated annealing algorithm and the pattern search method can accelerate the search and help the algorithm to obtain the optimal or near optimal solution in reasonable time.

5.5 The efficiency of applying the Nelder-Mead method in the proposed HSAPS algorithm with integer programming problems

The Nelder-Mead method is applied in the final stage of the proposed HSAPS algorithm in order to accelerate the convergence of the proposed algorithm and to avoid running the algorithm with more iterations without any improvement or slow convergence in the obtained results. The results in Table 4 show the mean evaluation function values of the proposed

Table 4: The efficiency of invoking the Nelder-Mead method in the final stage of HSAPS for $FI_1 - FI_7$ integer programming problems

Function	HSAPS without NM	HSAPS with NM
FI_1	548.23	**210.86**
FI_2	590.48	**199.12**
FI_3	2145.23	**637.48**
FI_4	589.45	**135.82**
FI_5	986.48	**624.08**
FI_6	389.98	**159.06**
FI_7	425.69	**140.08**

HSAPS without and with applying Nelder-Mead method, respectively. We report the best results in **boldface** text. The results in Table 4 show that invoking the Nelder-Mead method in the final stage can accelerate the search and help the algorithm to reach to the optimal or near optimal solution faster than the proposed algorithm without applying the Nelder-Mead method.

5.6 HSAPS and other algorithms

HSAPS is compared with other four benchmark algorithms (particle swarm optimization with different variants) in order to show the efficiency of the proposed algorithm. Before we discuss the comparison results of all algorithms, we present a brief description about the comparative four algorithms [38] as follows.

- **RWMPSOg**. RWMPSOg is a Random Walk Memetic Particle Swarm Optimization (with global variant), which combines the particle swarm optimization with random walk with direction exploitation.
- **RWMPSOl**. RWMPSOl is a Random Walk Memetic Particle Swarm Optimization (with local variant), which combines the particle swarm optimization with random walk with direction exploitation.
- **PSOg**. PSOg is a standard particle swarm optimization with global variant without local search method.
- **PSOl**. PSOl is a standard particle swarm optimization with local variant without local search method.

5.6.1 Comparison between RWMPSOg, RWMPSOl, PSOg, PSOl and HSAPS for integer programming problems.

In this subsection, we present the comparison results between our HSAPS algorithm and the other algorithms in order to verify the efficiency of our proposed algorithm. The five comparative algorithms are tested on 7 benchmark functions, which are mentioned in Subsection 5.2. The results of the other comparative algorithms are taken from their original papers [38]. The minimum (min), maximum (max), average (Mean), standard deviation (St.D) and Success rate (%Suc) of the evaluation function values are reported over 50 runs and reported in Table 5. The run is considered successful if the algorithm reached to the global minimum

of the solution within an error of 10^{-4} before the 20,000 function evaluation value. The best results between the comparative algorithms are reported with **boldface** text. The results in Table 5 show that the proposed HSAPS algorithm is successful in all runs and obtains the desired objective value of each function faster than the other algorithms in 6 of 7 test functions.

5.7 Simulated annealing heuristic pattern search (SAHPS) method and HSAPS for integer programming problems

In this subsection, we compare the HSAPS algorithm with another simulated annealing based method, which is called simulated annealing pattern search method (SAHPS) [17]. Before reporting the comparison results between our HSAPS algorithm and SAHPS method, we give a brief overview of SAHPS method as follows.

5.7.1 Simulated annealing heuristic pattern search (SAHPS)

SAHPS is a hybrid simulated annealing method proposed in [17] in order to solve multi-model functions by combining the simulated annealing, heuristic pattern search (HPS) and the Nelder-Mead method. The general structure of the SAHPS method looks like the general structure of our HSAPS, however, they are working in different way. In SAHPS, the SA trail solutions are generated along the direction sign between the current solution and another exploring solution generated close to the current solution. The applied heuristic pattern search (HPS) in SAHPS is a combination between the Approximate Descent Direction (ADD) method and the pattern search method. The HPS represents the intensification part in the SAHPS method, however it is not applied in each iteration in the epoch length (SA inner loop). It is applied only when the SA fails to obtain a good trail solution in the epoch length. On the other side we applied the pattern search method as a local search method at each iteration in the epoch length in order to accelerate the search and overcome of the slow convergence of the simulated annealing method.

5.7.2 Comparison between simulated annealing heuristic pattern search (SAHPS) method and HSAPS for integer programming problems

In Table 6, we give the comparison results between the SAHPS method and the proposed HSAPS. The results of the SAHPS method are taken after running the code of the method. The comparative results between the SAHPS method and the HSAPS algorithm are reported in Table 6. The average (Mean), standard deviation (St.D) and rate of success (Suc) are reported over 30 runs in Table 6. The best mean evaluation values between the two algorithms are reported with **boldface** text. The results in Table 6 show that the results of HSAPS algorithm are better than the results of the SAHPS method in all functions. The overall results in Table 6 show that the HSAPS algorithm is faster and more efficient than the SAHPS method.

5.8 HSAPS and the branch and bound method

Another investigated experiment has been applied in this paper to verify how the proposed algorithm is powerful by comparing the HSAPS algorithm against the branch and bound (BB) method [4], [5], [28], [31]. Before we discuss the comparative results between the proposed algorithm and the BB method, we present the BB method and the main steps of its algorithm as follows.

Table 5: Experimental results (min, max, mean, standard deviation and rate of success) of function evaluation for $FI_1 - FI_7$ test problems

Function	Algorithm	Min	Max	Mean	St.D	Suc
FI_1	RWMPSOg	17,160	74,699	27,176.3	8657	50
	RWMPSOl	24,870	35,265	30,923.9	2405	50
	PSOg	14,000	261,100	29,435.3	42,039	34
	PSOl	27,400	35,800	31,252	1818	50
	HSAPS	182	231	**210.86**	9.34	50
FI_2	RWMPSOg	252	912	578.5	136.5	50
	RWMPSOl	369	1931	773.9	285.5	50
	PSOg	400	1000	606.4	119	50
	PSOl	450	1470	830.2	206	50
	HSAPS	169	230	**199.12**	11.7	50
FI_3	RWMPSOg	361	41,593	6490.6	6913	50
	RWMPSOl	5003	15,833	9292.6	2444	50
	PSOg	2150	187,000	12,681	35,067	50
	PSOl	4650	22,650	11,320	3803	50
	HSAPS	545	709	**637.48**	47.06	50
FI_4	RWMPSOg	76	468	215	97.9	50
	RWMPSOl	73	620	218.7	115.3	50
	PSOg	100	620	369.6	113.2	50
	PSOl	120	920	390	134.6	50
	HSAPS	127	213	**135.82**	11.84	50
FI_5	RWMPSOg	687	2439	1521.8	360.7	50
	RWMPSOl	675	3863	2102.9	689.5	50
	PSOg	680	3440	1499	513.1	43
	PSOl	800	3880	2472.4	637.5	50
	HSAPS	519	681	**624.08**	624.08	44.58
FI_6	RWMPSOg	40	238	**110.9**	48.6	50
	RWMPSOl	40	235	112	48.7	50
	PSOg	80	350	204.8	62	50
	PSOl	70	520	256	**107.5**	50
	HSAPS	138	185	159.06	12.90	50
FI_7	RWMPSOg	72	620	242.7	132.2	50
	RWMPSOl	70	573	248.9	134.4	50
	PSOg	100	660	421.2	130.4	50
	PSOl	100	820	466	165	50
	HSAPS	112	169	**140.08**	15.06	50

Table 6: Experimental results (mean, standard deviation and rate of success) of function evaluation between SAHPS and HSAPS for $FI_1 - FI_7$ test problems

Function	Algorithm	Mean	St.D	Suc
FI_1	SAHPS	578.13	50.23	30
	HSAPS	**206**	7.033	30
FI_2	SAHPS	468.25	40.15	30
	HSAPS	**200.6**	10.39	30
FI_3	SAHPS	848.23	25.48	30
	HSAPS	**648.23**	45.35	30
FI_4	SAHPS	300.98	15.68	30
	HSAPS	**134.53**	7.23	30
FI_5	BB	754	10.30.1	30
	HSAPS	**616.1**	42.59	30
FI_6	SAHPS	434.66	78.69	30
	HSAPS	**162.16**	21.48	30
FI_7	SAHPS	309.96	58.69	30
	HSAPS	**145.06**	12.58	30

5.8.1 Branch and bound method

The branch and bound method (BB) is one of the most widely used method for solving optimization problems. The main idea of BB method is the feasible region of the problem is partitioned subsequently into several sub regions, this operation is called branching. The lower and upper bounds value of the function can be determined over these partitions, this operation is called bounding. The main steps of BB method are mentioned in Algorithm 6, and we can summarize the BB algorithm in the following steps.

Algorithm 6 The branch and bound algorithm

1: Set the feasible region M_0, $M_0 \supset S$
2: Set $i = 0$
3: **repeat**
4: Set $i = i + 1$
5: Partition the feasible region M_0 into many subsets Mi
6: For each subset M_i, determine lower bound β, where $\beta = \min\ \beta(M_i)$
7: For each subset M_i, determine upper bound α, where $\alpha = \min\ \alpha(M_i)$
8: **if** $(\alpha = \beta)||(\alpha - \beta \leq \epsilon)$ **then**
9: Stop
10: **else**
11: Select some of the subset M_i and partition them
12: **end if**
13: Determine new bound on the new partition elements
14: **until** $(i \leq m)$

Step 1. The algorithm starts with a relaxed feasible region $M_0 \supset S$, where S is the feasible region of the problem. This feasible region M_0 is partitioned into finitely many subsets M_i.

Table 7: Experimental results (mean, standard deviation and rate of success) of function evaluation between BB and HSAPS for $FI_1 - FI_7$ test problems

Function	Algorithm	Mean	St.D	Suc
FI_1	BB	1167.83	659.8	30
	HSAPS	**206**	7.033	30
FI_2	BB	**139.7**	102.6	30
	HSAPS	200.6	10.39	30
FI_3	BB	4185.5	32.8	30
	HSAPS	**648.23**	45.35	30
FI_4	BB	316.9	125.4	30
	HSAPS	**134.53**	7.23	30
FI_5	BB	2754	1030.1	30
	HSAPS	**616.1**	42.59	30
FI_6	BB	211	15	30
	HSAPS	**162.16**	21.48	30
FI_7	BB	358.6	14.7	30
	HSAPS	**145.06**	12.58	30

Step 2. For each subset M_i, the lower bound β and the upper bound α have been determined, where
$\beta(M_i) \leq \inf f(M_i \bigcap S) \leq \alpha(M_i)$, f is the objective function.

Step 3. The algorithm is terminated, if the bounds are equal or very close, i.e $\alpha = \beta$ (or $\alpha - \beta \leq \epsilon$), ϵ is a predefined positive constant.

Step 4. Otherwise, if the bounds are not equal or very close, some of the subsets M_i are selected and partitioned in order to obtain a more refined partition of M_0.

Step 5. The procedure is repeated until termination criteria are satisfied.

5.8.2 Comparison between the BB method and HSAPS for integer programming problems

In Table 7, we give the comparison results between the BB method and the proposed HSAPS. The results of the BB method are taken from [27]. In [27], the BB algorithm transforms the initial integer problem programming problem to a continues one. For the bounding, the BB uses the sequential quadratic programming method to solve the generated sub problems. While for branching, depth first traversal with backtracking was used. The comparative results between the BB algorithm and the proposed algorithm are reported in Table 7. The average (Mean), standard deviation (St.D) and rate of success (Suc) are reported over 30 runs in Table 7. The best mean evaluation values between the two algorithms are reported with **boldface** text. The results in Table 7 show that the proposed algorithm results are better than the results of the BB method in 6 of 7 functions. The overall results in Table 7 show that the proposed algorithm is faster and more efficient than the BB method.

The final conclusion after applying the proposed algorithm on the integer programming problems and comparing it with different 5 algorithms, that the proposed HSAPS algorithm is a promising algorithm and can obtain the optimal or near optimal solution of the integer programming functions faster than the other comparative algorithms.

5.9 Minimax problems

Test problems for other optimization problems have been selected in order to investigate the efficiency of the proposed algorithm. These functions contain 10 benchmark minimax functions, their properties are reported in Table 8 and the form of each function is listed as follows.

Test problem 1 ([50]). This problem is defined by

$$\begin{aligned} &\min\ FM_1(x), \\ &\qquad FM_1(x) = \quad \max\ f_i(x),\ \ i = 1,2,3, \\ &\qquad f_1(x) = \quad x_1^2 + x_2^4, \\ &\qquad f_2(x) = \quad (2-x1)^2 + (2-x_2)^2, \\ &\qquad f_3(x) = \quad 2exp(-x_1 + x_2) \end{aligned}$$

Test problem 2 ([50]). This problem is defined by

$$\begin{aligned} &\min\ FM_2(x), \\ &\qquad FM_2(x) = \quad \max\ f_i(x),\ \ i = 1,2,3, \\ &\qquad f_1(x) = \quad x_1^4 + x_2^2, \\ &\qquad f_2(x) = \quad (2-x1)^2 + (2-x_2)^2, \\ &\qquad f_3(x) = \quad 2exp(-x_1 + x_2) \end{aligned}$$

Test problem 3 ([50]). This problem is a nonlinear programming problem and transformed to minimax problem according to Equations 2.4, 2.5, and it is defined by

$$\begin{aligned} FM_3(x) = &\quad x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4, \\ g_2(x) = &\quad -x_1^2 - x_2^2 - x_3^3 - x_4^2 - x_1 + x_2 - x_3 + x_4 + 8, \\ g_3(x) = &\quad -x_1^2 - 2x_2^2 - x_3^2 - 2x_4 + x_1 + x_4 + 10, \\ g_4(x) = &\quad -x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5 \end{aligned}$$

Test problem 4 ([50]). This problem is a nonlinear programming problem and it is defined by

$$\begin{aligned} &\min\ FM_4(x), \\ &\qquad FM_4(x) = \quad \max f_i(x)\ \ i = 1,\ldots,5 \\ &\qquad f_1(x) = \quad (x_1-10)^2 + 5(x_2-12)^2 + x_3^4 + 3(x_4-11)^2 + \\ &\qquad\qquad +10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7, \\ &\qquad f_2(x) = \quad f_1(x) + 10(2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 - 127), \\ &\qquad f_3(x) = \quad f_1(x) + 10(7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 - 282), \\ &\qquad f_4(x) = \quad f_1(x) + 10(23x_1 + x_2^2 + 6x_6^2 - 8x_7 - 196), \\ &\qquad f_5(x) = \quad f_1(x) + 10(4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 \end{aligned}$$

Test problem 5 ([44]). This problem is defined by

$$\begin{aligned} &\min\ FM_5(x), \\ &\qquad FM_5(x) = \quad \max\ f_i(x),\ \ i = 1,2, \\ &\qquad f_1(x) = \quad |x_1 + 2x_2 - 7|, \\ &\qquad f_2(x) = \quad |2x_1 + x_2 - 5| \end{aligned}$$

Test problem 6 ([44]). This problem is defined by

$$\begin{aligned} \min\ & FM_6(x), \\ & FM_6(x) = \max\ f_i(x), \\ & f_i(x) = |x_i|,\ \ i = 1, \ldots, 10, \end{aligned}$$

Test problem 7 ([30]). This problem is defined by

$$\begin{aligned} \min\ & FM_7(x), \\ & FM_7(x) = \max\ f_i(x),\ \ i = 1, 2, \\ & f_1(x) = (x_1 - \sqrt{(x_1^2 + x_2^2)} cos \sqrt{x_1^2 + x_2^2})^2 + 0.005(x_1^2 + x_2^2)^2, \\ & f_2(x) = (x_2 - \sqrt{(x_1^2 + x_2^2)} sin \sqrt{x_1^2 + x_2^2})^2 + 0.005(x_1^2 + x_2^2)^2 \end{aligned}$$

Test problem 8 ([30]). This problem is defined by

$$\begin{aligned} \min\ & FM_8(x), \\ & FM_8(x) = \max\ f_i(x),\ \ i = 1, \ldots, 4, \\ & f_1(x) = \big(x_1 - (x_4 + 1)^4\big)^2 + \Big(x_2 - \big(x_1 - (x_4 + 1)^4\big)^4\Big)^2 + \\ & \quad 2x_3^2 + x_4^2 - 5\big(x_1 - (x_4 + 1)^4\big) - 5\Big(x_2 - \big(x1 - \\ & \quad (x_4 + 1)^4\big)^4\Big) - 21x_3 + 7x_4, \\ & f_2(x) = f_1(x) + 10\Big[\big(x_1 - (x_4 + 1)^4\big)^2 + \Big(x_2 - \big(x_1 - \\ & \quad (x_4 + 1)^4\big)^4\Big)^2 + x_3^2 + x_4^2 + \big(x_1 - (x_4 + 1)^4\big) - \\ & \quad \Big(x_2 - \big(x_1 - (x_4 + 1)^4\big)^4\Big) + x_3 - x_4 - 8\Big], \\ & f_3(x) = f_1(x) + 10\Big[\big(x_1 - (x_4 + 1)^4\big)^2 + 2\Big(x_2 - \big(x_1 - \\ & \quad (x_4 + 1)^4\big)^4\Big)^2 + x_3^2 + 2x_4^2 - \big(x_1 - (x_4 + 1)^4\big) - \\ & \quad x_4 - 10\Big], \\ & f_4(x) = f_1(x) + 10\Big[\big(x_1 - (x_4 + 1)^4\big)^2 + \Big(x_2 - \big(x_1 - \\ & \quad (x_4 + 1)^4\big)^4\Big)^2 + x_3^2 + 2\big(x_1 - (x_4 + 1)^4\big) - \\ & \quad \Big(x_2 - \big(x_1 - (x_4 + 1)^4\big)^4\Big) - x_4 - 5\Big], \end{aligned} \tag{5.1}$$

Test problem 9 ([30]). This problem is a nonlinear programming problem and trans-

Table 8: Minimax test functions properties.

Function	Dimension (d)	Desired error goal
FM_1	2	1.95222245
FM_2	2	2
FM_3	4	-40.1
FM_4	7	247
FM_5	2	10^{-4}
FM_6	10	10^{-4}
FM_7	2	10^{-4}
FM_8	4	-40.1
FM_9	7	680
FM_{10}	4	0.1

formed to minimax problem according to Equations 2.4, 2.5, and it is defined by

$$\begin{aligned}
&\min \ FM_9(x), \\
&\qquad FM_9(x) = \ \max \ f_i(x), \ \ i = 1, \ldots, 5, \\
&\qquad f_1(x) = \ (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 \\
&\qquad\qquad +10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7, \\
&\qquad f_2(x) = \ -2x_1^2 - 2x_3^4 - x_3 - 4x_4^2 - 5x_5 + 127, \\
&\qquad f_3(x) = \ -7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 + 282, \\
&\qquad f_4(x) = \ -23x_1 - x_2^2 - 6x_6^2 + 8x_7 + 196, \\
&\qquad f_5(x) = \ -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7
\end{aligned}$$

Test problem 10 ([30]). This problem is defined by

$$\begin{aligned}
&\min \ FM_{10}(x), \\
&\qquad FM_{10}(x) = \ \max|f_i(x)|, \quad i = 1, \ldots, 21, \\
&\qquad f_i(x) = \ x_1 exp(x_3 t_i) + x_2 exp(x_4 t_i) - \frac{1}{1 + t_i}, \\
&\qquad t_i = \ -0.5 + \frac{i - 1}{20}
\end{aligned}$$

5.10 The efficiency of the proposed HSAPS algorithm with minimax problems

We will start to investigate the efficiency of combining the simulated annealing algorithm with the pattern search method to solve minimax problems. This test is applied without invoking the final intensification process (Nelder-Mead method). The parameter setting values for both algorithms ware the same for both algorithms in order to make a fair comparison. The functions FM_2, FM_3, FM_4, FM_7 and FM_9 have been selected to show the efficiency of the proposed algorithm by plotting the values of function values versus the number of iterations as shown in Figure 4. In Figure 4, the solid line refers to the proposed HSAPS results, while the dotted line refers to the standard simulated annealing algorithm results after 50 iterations. Figure 4 shows that the function values are rapidly decreases as the number of iterations increases for HSAPS results than those of the standard simulated annealing

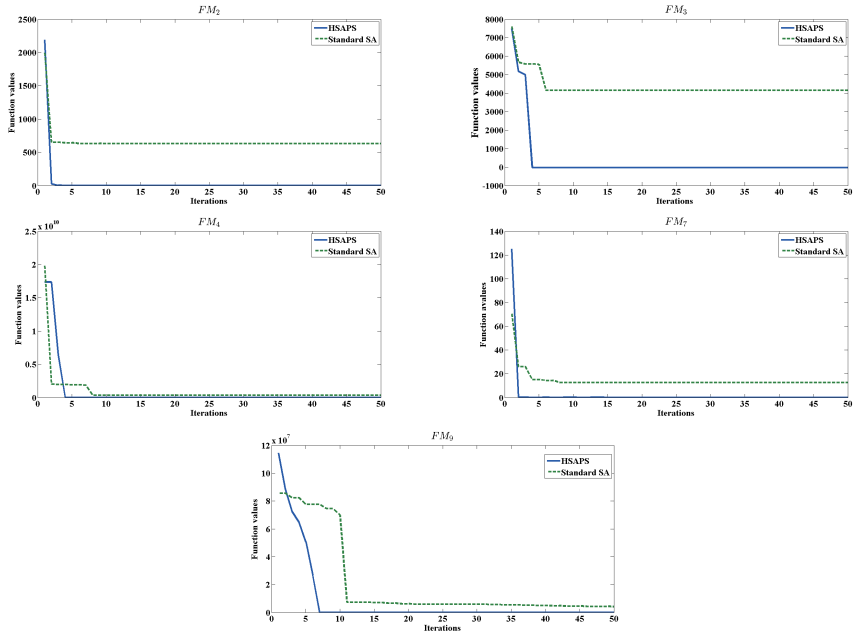

Figure 4: The efficiency of the proposed HSAPS algorithm with minimax problems

algorithm. The results in Figure 4 show that the combination between the standard simulated annealing algorithm and the pattern search method can improve the performance of the standard simulated annealing algorithm and accelerate the convergence of the proposed algorithm.

5.11 The general performance of the HSAPS algorithm with minimax problems

The proposed HSAPS have been investigated to verify of the general performance of it with the minimax problems by plotting the values of function values versus the number of iterations as shown in Figure 5 for five test functions FM_1, FM_5, FM_6, FM_8 and FM_{10}. The results in Figure 5 are the results of the proposed algorithm without applying the Nelder-Mead method in the final stage of the algorithm after 50 iterations. The results in Figure 5 show that the function values of the proposed HSAPS are rapidly decreases as the number of iterations increases which verified that the hybridization between the simulated annealing algorithm and the pattern search method can accelerate the search and helps the algorithm to obtain the optimal or near optimal solution in few iterations only.

5.12 The efficiency of applying the Nelder-Mead method in the proposed HSAPS algorithm with minimax problems

The final test which we have applied in order to investigate the general performance of the proposed algorithm is to verify the importance of invoking the Nelder-Mead method in the final stage as a final intensification process. The results in Table 9 show the mean evaluation function values of the proposed HSAPS without and with applying Nelder-Mead method

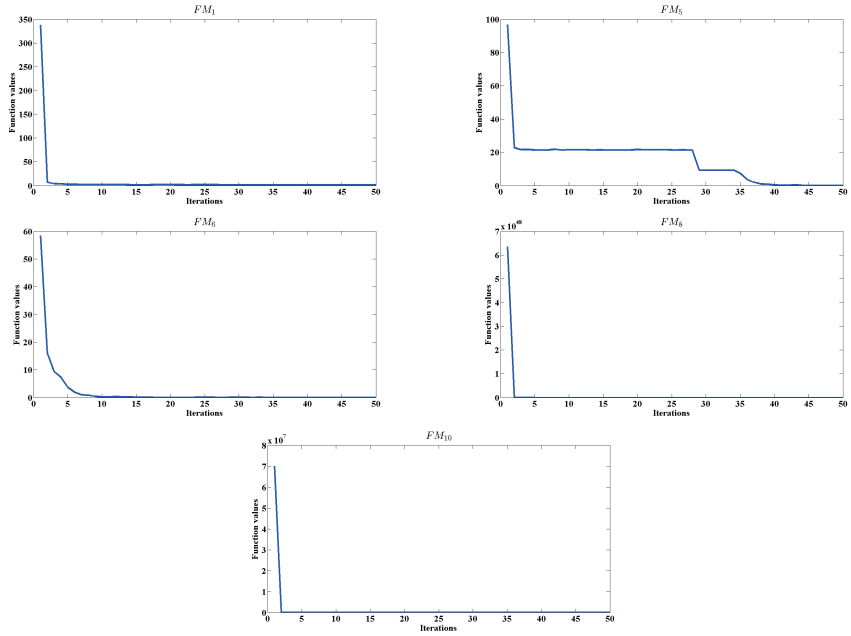

Figure 5: The general performance of HSAPS algorithm with minimax problems

respectively. The best results are reported in **boldface** text. The results in Table 9, show that invoking the Nelder-Mead method in the final stage enhance the general performance of the proposed algorithm and can accelerate the search to reach to the optimal or near optimal solution faster than the proposed algorithm without applying the Nelder-Mead method.

5.13 HSAPS and other algorithms

HSAPS is compared with three benchmark algorithms in order to verify of the efficiency of the proposed algorithm with minimax problems. Before discussing the comparison results of all algorithms, we present a brief description about the comparative three algorithms as follows.

- HPS2 [21]. HPS2 is a Heuristic Pattern Search algorithm, which is applied for solving bound constrained minimax problems by combining the Hook and Jeeves (HJ) pattern and exploratory moves with a randomly generated approximate descent direction.

- UPSOm [37]. UPSOm is a Unified Particle Swarm Optimization algorithm, which combines the global and local variants of the standard PSO and incorporates a stochastic parameter to imitate mutation in evolutionary algorithms.

- RWMPSOg [38]. RWMPSOg is a Random Walk Memetic Particle Swarm Optimization (with global variant), which combines the particle swarm optimization with random walk with direction exploitation.

Table 9: The efficiency of invoking the Nelder-Mead method in the final stage of HSAPS for $FM_1 - FM_{10}$ minimax problems

Function	HSAPS without NM	HSAPS with NM
FM_1	687.25	**205.05**
FM_2	489.56	**187.24**
FM_3	758.36	**454.36**
FM_4	873.59	**583.91**
FM_5	458.27	**127.52**
FM_6	548.38	**161.75**
FM_7	754.54	**493.14**
FM_8	20,000	**1754.45**
FM_9	724.85	**597.22**
FM_{10}	874.25	**486.15**

5.13.1 Comparison between HPS2, UPSOm, RWMPSOg and HSAPS for minimax problems

In this subsection, we present the comparison results between our HSAPS algorithm and the other mentioned algorithms in order to verify of the efficiency of the proposed algorithm. The four comparative algorithms are tested on 10 benchmark functions, which are reported in Subsection 5.9. The results of the comparative algorithms are taken from there original papers [21]. The average (Avg), standard deviation (SD) and Success rate (%Suc) are reported over 100 runs and reported in Table 10. The mark (-) for FM_8 in HPS2 algorithm and FM_2, FM_8 and FM_9 in RWMPSOg algorithm in Table 10 means that the results of these algorithms for that functions are not reported in its original papers. The run is considered succussed if the algorithm reached the global minimum of the solution within an error of 10^{-4} before the 20,000 function evaluation value. The results in Table 10, show that the proposed HSAPS algorithm is succussed in all runs and obtains the objective value of each function faster than the other algorithms, except for functions FM_3, FM_9 and FM_{10}.

5.14 Simulated annealing heuristic pattern search (SAHPS) method and HSAPS for minimax problems

The SAHPS method, has been applied on the minimax problems in order to compare its results against our proposed HSAPS algorithm. The results of the comparison are reported in Table 11 as follows.

5.14.1 Comparison between the SAHPS method and HSAPS for minimax problems

Another experiment has been done in order to compare our HSAPS algorithm and the SAHPS method to solve minimax problems. The results of the comparison are reported in Table 11. The average (Mean), standard deviation (St.D) and rate of success (Suc) are reported over 30 runs in Table 11. The best mean evaluation values between the two algorithms are reported in **boldface** text. The results in Table 11 show that the HSAPS algorithm results are better than the results of the SAHPS method in all functions. The

Table 10: Evaluation function for the minimax problems $FM_1 - FM_{10}$

Algorithm	Problem	Avg	SD	%Suc
HPS2	FM_1	1848.7	2619.4	99
	FM_2	635.8	114.3	94
	FM_3	**141.2**	**28.4**	37
	FM_4	8948.4	5365.4	7
	FM_5	772.0	60.8	100
	FM_6	1809.1	2750.3	94
	FM_7	4114.7	1150.2	100
	FM_8	-	-	-
	FM_9	**283.0**	**123.9**	64
	FM_{10}	**324.1**	173.1	100
UPSOm	FM_1	1993.8	853.7	100
	FM_2	1775.6	241.9	100
	FM_3	1670.4	530.6	100
	FM_4	12,801.5	5072.1	100
	FM_5	1701.6	184.9	100
	FM_6	18,294.5	2389.4	100
	FM_7	3435.5	1487.6	100
	FM_8	6618.50	2597.54	100
	FM_9	2128.5	597.4	100
	FM_{10}	3332.5	1775.4	100
RWMPSOg	FM_1	2415.3	1244.2	100
	FM_2	-	-	-
	FM_3	3991.3	2545.2	100
	FM_4	7021.3	1241.4	100
	FM_5	2947.8	257.0	100
	FM_6	18,520.1	776.9	100
	FM_7	1308.8	505.5	100
	FM_8	-	-	-
	FM_9	-	-	-
	FM_{10}	4404.0	3308.9	100
HSAPS	FM_1	**215.05**	7.53	100
	FM_2	**195.14**	10.04	100
	FM_3	472.32	99.87	100
	FM_4	**581.91**	7.95	100
	FM_5	**120.72**	20.16	100
	FM_6	**157.93**	127.86	100
	FM_7	**485.74**	116.53	100
	FM_8	**1535.36**	32.49	5
	FM_9	584.4	3.95	7
	FM_{10}	400.15	119.62	60

Table 11: Experimental results (mean, standard deviation and rate of success) of function evaluation between SQP and HSAPS for $FM_1 - FM_{10}$ test problems

Function	Algorithm	Mean	St.D	Suc
FM_1	SAHPS	313.93	23.48	30
	HSAPS	**205.05**	11.83	30
FM_2	SAHPS	319.47	15.48	30
	HSAPS	**187.24**	17.04	30
FM_3	SAHPS	472.54	20.18	30
	HSAPS	**454.36**	102.96	30
FM_4	SAHPS	1715.48	116.45	10
	HSAPS	**583.91**	8.24	25
FM_5	SAHPS	315.48	17.25	30
	HSAPS	**127.52**	25.6	30
FM_6	SAHPS	20,000	0.0	0.0
	HSAPS	**161.75**	131.15	30
FM_7	SAHPS	687.25	52.47	10
	HSAPS	**493.14**	119.29	30
FM_8	SAHPS	1147	95.48	10
	HSAPS	**1014.45**	57.12	3
FM_9	SAHPS	758.24	158.47	4
	HSAPS	**597.22**	13.65	5
FM_{10}	SAHPS	649.78	58.47	15
	HSAPS	**486.15**	123.14	10

overall results in Table 11 show that the HSAPS algorithm is faster and more efficient than the SAHPS method.

5.15 HSAPS and SQP method

The last test for our proposed algorithm is to compare the HSAPS with another famous method which is called sequential quadratic programming method (SQP). In the following subsection, we highlight the main steps of the SQP method and how it works.

5.15.1 Sequential quadratic programming (SQP)

The first references to SQP algorithms have been in Wilson's PhD thesis in 1963 [49], he proposed the Newton-SQP algorithm to solve unconstrained optimization. The development of the secant or variable-metric algorithms led to the extension of these methods to solve the constrained problem. The main steps of the SPQ method can be summarized as follows.

Step 1. The SQP algorithm starts with an initial solution x_0, the Hessian matrix of the objective function is initialized.

Step 2. At each iteration, the BFGS method has been used to calculate a positive definite quasi-Newton approximation of the Hessian matrix , where the Hessian update is calculated as follows

$$H_{n+1} = H_n + \frac{q_n q_n^T}{q_n^T s_n} - \frac{H_n^T H_n}{s_n^T H_n s_n}, \tag{5.2}$$

Table 12: Experimental results (mean, standard deviation and rate of success) of function evaluation between SQP and HSAPS for $FM_1 - FM_{10}$ test problems

Function	Algorithm	Mean	St.D	Suc
FM_1	SQP	4044.5	8116.6	24
	HSAPS	**205.05**	11.83	30
FM_2	SQP	8035.7	9939.9	18
	HSAPS	**187.24**	17.04	30
FM_3	SQP	**135.5**	21.1	30
	HSAPS	454.36	102.96	30
FM_4	SQP	20,000	0.0	0.0
	HSAPS	**583.91**	8.24	30
FM_5	SQP	140.6	38.5	30
	HSAPS	**127.52**	25.6	30
FM_6	SQP	611.6	200.6	30
	HSAPS	**161.75**	131.15	30
FM_7	SQP	15,684.0	7302.0	10
	HSAPS	**493.14**	119.29	30
FM_8	SQP	20,000	0.0	0.0
	HSAPS	**1754.45**	57.12	3
FM_9	SQP	20,000	0.0	0.0
	HSAPS	**597.22**	13.65	5
FM_{10}	SQP	4886.5	8488.4	22
	HSAPS	**486.15**	123.14	50

where $s_n = x_{n+1} - x_n$ and $q_n = \nabla f(x_{n+1})$

Step 3. The QP problem is solved in z as follows

$$min\ q(z) = 1/2 z^T H z + c^T z. \tag{5.3}$$

Step 4. The new potential solution is calculating using the solution z_n as follows

$$x_{n+1} = x_n + \alpha_n z_n \tag{5.4}$$

where α_n is a step length and determined through line search.

For an extended theoretical aspects of the SQP algorithm refer to [10], [12].

The results of the two comparative algorithms are tested on 10 benchmark functions, which are reported in Subsection 5.9. The results of the SQP algorithm are reported and taken from its paper [27]. The average (Avg), standard deviation (SD) and Success rate (%Suc) are reported over 30 runs and reported in Table 12. The run is considered successed if the algorithm reached the global minimum of the solution within an error of 10^{-4} before the 20,000 function evaluation value. The results in Table 12, show that the proposed HSAPS algorithm is outperform the SQP algorithm in all functions, except FM_3. The final conclusion from this comparison is that the proposed HSAPS is outperform the SQP algorithm in most of tested minimax problems.

6 Conclusion

In this paper, a new hybrid algorithm has been proposed by combining the simulated annealing algorithm with the pattern search and the Nelder Mead methods in order to solve integer programming and minimmax problems. The proposed algorithm is called hybrid simulated annealing and pattern search (HSAPS) algorithm. In the proposed algorithm, we try to overcome the main drawback of the simulated annealing algorithm by accelerating the search through invoking the pattern search and Nelder-Mead methods in the standard simulated annealing algorithm. The pattern search method is working as a local search method and it used to refine the simulated annealing solution at each iteration, while the Nelder-Mead method is used in the final stage of the algorithm in order to refine the overall best solution and to avoid running the algorithm more iteration without any improvement in the search. The HSAPS algorithm has been intensely tested on 17 benchmark functions 7 integer programming problems and 10 minimax problems. The proposed algorithm is compared against other 6 algorithms to investigate its performance solving integer programming problems and 5 algorithms to test its performance for solving minimax problems. The numerical results indicate that the proposed HSAPS algorithm is a promising algorithm and suitable to find a global optimal solution or near optimal solution of the tested functions with their different properties in reasonable time.

Acknowledgments

We are grateful to the anonymous reviewers for constructive feedback and insightful suggestions which greatly improved this article.

References

[1] N. Bacanin and M. Tuba, Artificial bee colony (ABC) algorithm for constrained optimization improved with genetic operators, *Studies in Informatics and Control* 21 (2012) 137–146.

[2] N. Bacanin, I. Brajevic and M. Tuba, Firefly algorithm applied to integer programming problems, Recent Advances in Mathematics, 2013

[3] J.W. Bandler and C. Charalambous, Nonlinear programming using minimax techniques, *J. Optim. Theory Appl.* **13** (1974) 607–619.

[4] B. Borchers and J.E. Mitchell, *Using an Interior point method in a branch and bound algorithm for integer programming*, Technical Report, Rensselaer Polytechnic Institute, July 1992.

[5] B. Borchers and J.E. Mitchell, An improved branch and bound algorithm for mixed integer nonlinear programs, *Computers & Operations Research* 21 (1994) 359–367.

[6] M.F. Cardoso, R.L. Salcedo and S.F. de Azevedo ,The simplex-simulated annealing approach to continuous non-linear optimization, *Comput. Chem. Eng.* 20 (1996) 1065–1080.

[7] M.F. Cardoso, R.L. Salcedo, S.F. de Azevedo and D. Barbosa, A simulated annealing approach to the solution of minlp problems, *Comput. Chem. Eng.* 21 (1997) 1349–1364.

[8] D.Z. Du and P.M. Pardalose, *Minimax and Applications*, Kluwer Academic Publishers, Dordrecht, 1995.

[9] M. Dorigo, Optimization, learning and natural algorithms, Ph.D. Thesis, Politecnico di Milano, Italy, 1992.

[10] R. Fletcher, *Practical Method of Optimization*, Vol.1 & 2, John Wiley and Sons, 1980.

[11] M.J. Flynn, Some computer organizations and their effectiveness, IEEE Transon Computers **C-21** (1972) 948–960.

[12] P. E. Gill, W. Murray and M.H. Wright, *Practical Optimization*, Academic Press, London, 1981.

[13] A. Glankwahmdee, J.S. Liebman and G.L. Hogg, Unconstrained discrete nonlinear programming, *Engineering Optimization* 4 (1979) 95–107.

[14] F. Glover, Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research* 13 (1986) 533–549.

[15] F. Glover, A template for scatter search and path relinking, in *Artificial Evolution*, Lecture Notes in Computer Science, vol. 1363, 1998, Springer, pp. 1–51,

[16] A. Hedar and M. Fukushima, Hybrid simulated annealing and direct search method for nonlinear unconstrained global optimization, *Optimization Methods and Software* 17 (2002) 891–912.

[17] A. Hedar and M. Fukushima, Heuristic pattern search and its hybridization with simulated annealing for nonlinear global optimization, *Optimization Methods and Software* 19 (2002) 891–912.

[18] F.S. Hillier and G.J. Lieberman, *Introduction to Operations Research*, 9th ed., McGraw-Hill, New York, 2005.

[19] J.H. Holland, —it Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor 1975.

[20] R. Hooke and T. A.Jeeves, Direct search , solution of numerical and statistical problems, *J. Assoc. Comput. Mach.* (1961) 212–229.

[21] A.C.P. Isabel, E. Santo and E. Fernandes, Heuristics pattern search for bound constrained minimax problems, in *Computational Science and its Applications*, Lecture Notes in Computer Science, Vol. 6784, ICCSA 2011, pp. 174–184.

[22] G.K. Jati and S. Suyanto, Evolutionary discrete firefly algorithm for travelling salesman problem, in *Adaptive and Intelligent Systems*, Lecture Notes in Computer Science, Vol. 6943, Springer, 2011, pp. 393–403.

[23] R. Jovanovic and M. Tuba, An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem, *Applied Soft Computing* 11 (2011) 5360–5366.

[24] R. Jovanovic, M. Tuba, Ant colony optimization algorithm with pheromone correction strategy for minimum connected dominating set problem, *Computer Science and Information Systems* 10 (2013) 133–149.

[25] S. Kirkpatrick, C.D. Gelatt and M.P. Vecchi, Optimization by simulated annealing, *Science* 220 (1983) 671–680.

[26] J. Kennedy and R.C. Eberhart, Particle swarm optimization, in *Proceedings of the IEEE International Conference on Neural Networks*, vol 4, 1995, pp. 194–1948.

[27] E.C. Laskari, K.E. Parsopoulos and M.N. Vrahatis, Particle swarm optimization for integer programming, in *Proceedings of the IEEE 2002 Congress on Evolutionary Computation*, Honolulu (HI), 2002, pp. 1582–1587.

[28] E.L. Lawler and D.W. Wood, Branch and bound methods: a survey, *Operations Research* 14 (1966) 699–719.

[29] G. Liuzzi, S. Lucidi and M. Sciandrone, A derivative-free algorithm for linearly constrained finite minimax problems, *SIAM J. Optim.* 16 (2006) 1054–1075.

[30] L. Lukšan and J. Vlček, Test problems for nonsmooth unconstrained and linearly constrained optimization, Technical report 798, Institute of Computer Science, Academy of Sciences of the Czech Republic, Prague, Czech Republic, 2000.

[31] V.M. Manquinho, J.P. Marques Silva, A.L. Oliveira ans K. A. Sakallah, Branch and bound algorithms for highly constrained integer programs, Technical Report, Cadence European Laboratories, Portugal, 1997.

[32] N. Mladenovic, Avariable neighborhood algorithm –a new metaheuristic for combinatorial optimization, in *Abstracts of Papers Presented at Optimization Days*, Montréal,Canada, 1995, pp. 112.

[33] M. Mladenovic and P. Hansen, Variable neighborhood search, *Computers and Operations Research* 24 (1997) 1097—1100.

[34] J. A. Nelder and R. Mead, A simplex method for function minimization, *Computer Journal* 7 (1965) 308–313.

[35] I.H. Osman and J.P. Kelly, *Meta-Heuristics: Theory and Applications*, Kluwer Academic Publishers, Boston, MA, 1996.

[36] G. L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd (eds.), *Handbooks in OR & MS*, volume 1. Elsevier, 1989.

[37] K.E. Parsopoulos and M.N. Vrahatis, Unified particle swarm optimization for tackling operations research problems, in *Proceeding of IEEE 2005 swarm Intelligence Symposium*, Pasadena, USA, 2005, pp. 53–59.

[38] Y.G. Petalas, K.E. Parsopoulos and M.N. Vrahatis, Memetic particle swarm optimization, *Ann oper Res* 156 (2007) 99–127.

[39] E. Polak, J.O. Royset and R.S. Womersley, Algorithms with adaptive smoothing for finite minimax problems, *J. Optim. Theory Appl.* 119 (2003) 459–484.

[40] S.S. Rao, *Engineering Optimization-Yheory and Practice*, Wiley, New Delhi, 1994.

[41] C. C. Ribeiro and P. Hansen (eds.), *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, Boston, MA, 2002.

[42] G. Rudolph, An evolutionary algorithm for integer programming, in *Parallel Problem Solving from Nature*, Davidor Y, Schwefel H-P, Männer R (eds.), vol. 3, 1994, pp. 139–148.

[43] Y. Saab and V. Rao, Combinatorial optimization by stochastic evolution, *IEEE Transactions on Computer-Aided Design* 10 (1991) 525–535.

[44] H.P. Schwefel, *Evolution and Optimum Seeking*, Wiley, New York, 1995.

[45] T. Stützle, Local search algorithms for combinatorial problems: Analysis, improvements, and new applications, Ph.D. Thesis, Darmstadt University of Technology, 1998.

[46] D. Teodorovic and M. DellOrco. Bee colony optimization cooperative learning approach to complex tranportation problems, in *Advanced OR and AI Methods in Transportation: Proceedings of 16th MiniEURO Conference and 10th Meeting of EWGT 2005*, Poznan: Publishing House of the Polish Operational and System Research, 2005, pp. 51–60.

[47] M. Tuba, N. Bacanin and N. Stanarevic, Adjusted artificial bee colony (ABC) algorithm for engineering problems, *WSEAS Transaction on Computers* 11 (2012) 111–120.

[48] M. Tuba, M. Subotic and N. Stanarevic, Performance of a modified cuckoo search algorithm for unconstrained optimization problems, *WSEAS Transactions on Systems* 11 (2012) 62–74.

[49] B. Wilson, A simplicial Algorithm for Concave Programming, PhD thesis, Harvard University, 1963.

[50] S. Xu, Smoothing method for minimax problems, *Computational Optimization and Applications* 20 (2001) 267–279.

[51] S. Zuhe, A. Neumaier and M.C. Eiermann, Solving minimax problems by interval methods, *BIT* 30 (1990) 742–751.

Manuscript received 27 February 2015
revised 12 July 2015
accepted for publication 13 July 2015

Ahmed F. Ali
Department of Computer Science, Faculty of Computers & Informatics
Suez Canal University, Ismailia, Egypt

Department of Mathematics and Statistics
Faculty of Science, Thompson Rivers University, Kamloops, BC, Canada V2C 0C8
E-mail address: ahmed_fouad@ci.suez.edu.eg

Mohamed A. Tawhid
Department of Mathematics and Statistics, Faculty of Science
Thompson Rivers University, Kamloops, BC, Canada V2C 0C8

Department of Mathematics and Computer Science
Faculty of Science, Alexandria University
Moharam Bey 21511, Alexandria, Egypt
E-mail address: Mtawhid@tru.ca