East Asian Journal on Applied Mathematics
doi: 10.4208/eajam.030915.210416a

Vol. **6**, No. 3, pp. 290-313
August 2016

Direct Gravitational Search Algorithm for Global Optimisation Problems

Ahmed F. Ali[1,2] and Mohamed A. Tawhid[2,3,*]

[1] *Department of Computer Science, Faculty of Computers & Informatics, Suez Canal University, Ismailia, Egypt.*
[2] *Department of Mathematics and Statistics, Faculty of Science, Thompson Rivers University, Kamloops, BC, Canada V2C 0C8.*
[3] *Department of Mathematics and Computer Science, Faculty of Science, Alexandria University, Moharam Bey 21511, Alexandria, Egypt.*

Received 3 September 2015; Accepted (in revised version) 21 April 2016.

Abstract. A gravitational search algorithm (GSA) is a meta-heuristic development that is modelled on the Newtonian law of gravity and mass interaction. Here we propose a new hybrid algorithm called the Direct Gravitational Search Algorithm (DGSA), which combines a GSA that can perform a wide exploration and deep exploitation with the Nelder-Mead method, as a promising direct method capable of an intensification search. The main drawback of a meta-heuristic algorithm is slow convergence, but in our DGSA the standard GSA is run for a number of iterations before the best solution obtained is passed to the Nelder-Mead method to refine it and avoid running iterations that provide negligible further improvement. We test the DGSA on 7 benchmark integer functions and 10 benchmark minimax functions to compare the performance against 9 other algorithms, and the numerical results show the optimal or near optimal solution is obtained faster.

AMS subject classifications: 49K35, 90C10, 68U20, 68W05

Key words: Gravitational search algorithm, direct search methods, Nelder-Mead method, integer programming problems, minimax problems.

1. Introduction

A gravitational search algorithm (GSA) is a population based meta-heuristic algorithm developed by Rashedi [32]. This and many other such meta-heuristic algorithms (including ant colony optimisation (ACO) [7], artificial bee colony [16], particle swarm optimisation (PSO) [17], bacterial foraging [28], bat algorithm [42], bee colony optimisation (BCO) [36], wolf search [35], cat swarm [6], firefly algorithm [41], and fish swarm/school

*Corresponding author. *Email addresses:* ahmed_fouad@ci.suez.edu.eg (A. F. Ali), Mtawhid@tru.ca (M. A. Tawhid)

 ©2016 Global-Science Press

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

[20]) have been used to solve unconstrained and constrained optimisation problems and their applications, and sometimes to minimax and integer programming problems.

Integer programming and combinatorial optimisation problems optimise functions of many variables subject to some problem specific constraints and integrality restrictions imposed on all or some of the variables, with many real applications such as airline crew scheduling, production planning, Internet routing, VLSI (very large scale integration), and packing and cutting. A combinatorial optimisation problem can often be modelled as an integer program [25, 26], but nevertheless these problems can be very difficult to solve as they are NP-hard [10]. Given their difficulty but practical importance, a large number of solution techniques for NP-hard integer and combinatorial optimisation problems have been proposed.

The available algorithms can be classified into two main classes — viz. exact and approximate algorithms. One of the most famous exact integer programming algorithms is Branch and Bound (BB), but suffers from high complexity in solving large scale problems as it explores hundreds of nodes in big tree structures. Recently, approximate algorithms such as swarm intelligence algorithms (ant colony algorithm [14, 15], artificial bee colony algorithm [1, 37], particle swarm optimisation algorithm [29], cuckoo search algorithm [38] and firefly algorithm [3]) have been used to solve integer programming problems

Minimax problems appear in many engineering areas such as optimal control, engineering design, discrete optimisation, Chebyshev approximation, game theory, computer-aided design, and circuit design — e.g. see Ref. [8, 43]. Moreover, any nonlinear programming problem with nonlinear constraints can be transformed into an equivalent unconstrained minimax problem [2]. There are some algorithms to solve minimax problems, involving the solution of a sequence of smoothing problems that approximate the minimax problems in the limit [21, 30, 40]. Thus these algorithms generate a sequence of approximations, which converges to the Kuhn-Tucker point of the given minimax problem for a decreasing sequence of positive smoothing parameters. However, a drawback is that these parameters can become small too quickly such that the smooth problems become significantly ill-conditioned. Some swarm intelligence algorithms have been applied to solve minimax problems (e.g. the PSO [29]), but their main drawback in solving minimax and integer programming problems is that they are computationally expensive.

We propose a new hybrid gravitational search algorithm and Nelder-Mead method called the Direct Gravitational Search Algorithm (DGSA), to overcome the slow convergence of the standard gravitational search algorithm (GSA) for solving global optimisation problems. The Nelder-Mead direct search method can accelerate the search and improve the convergence, avoiding additional iterations that provide negligible further improvement. The integer programming, minimax problems and the Nelder-Mead method are presented in Section 2. Section 3 summarises the main concepts of a gravitational search algorithm (GSA). The main structure of the proposed DGSA is presented in Section 4, and then tested on 7 integer programming and 10 minimax benchmark problems. The experimental results in Section 5 show that the DSGA can obtain the optimal or near optimal solution in most cases, and our concluding remarks are in Section 6.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

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

We first summarise aspects of integer programming and minimax problems.

2.1. The integer programming problem

An integer programming problem is a mathematical optimisation problem where all of the variables are integers. The unconstrained integer programming problem can be defined as follows.

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

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

2.2. Minimax problem

The general form of the minimax problem is [40]

$$\begin{aligned} &\min \ F(x)\,, \\ &F(x) = \max \ f_i(x)\,, \quad i = 1, \cdots, m\,, \end{aligned}$$

where $f_i(x) : S \subset \mathbb{R}^n \to \mathbb{R}$, $i = 1, \cdots, m$.

A nonlinear programming problem of the form

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

can be transformed to a minimax problem

$$\begin{aligned} &\min \ \max \ f_i(x)\,, \quad i = 1, \cdots, m\,, \\ &f_1(x) = F(x)\,, \\ &f_i(x) = F(x) - \alpha_i g_i(x)\,, \\ &\alpha_i > 0\,, \quad i = 2, \cdots, m\,. \end{aligned}$$

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 [2].

2.3. Nelder Mead method

In 1965, Nelder and Mead [24] proposed the Nelder-Mead algorithm (NM), one of the most popular derivative-free nonlinear optimisation algorithms. It starts with $n+1$ points (vertices) $x_1, x_2, \cdots, x_{n+1}$ that are evaluated, ordered and re-labeled in order to assign the best and the worst points. In minimisation problems, x_1 is considered to be the best vertex point if it has the minimum value of the objective function, while the worst point x_{n+1} has

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

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 — viz. the coefficients of reflection ρ, expansion χ, contraction τ and shrinkage ϕ (where $\rho > 0$, $\chi > 1$, $0 < \tau < 1$, and $0 < \phi < 1$). The main steps of the Nelder-Mead algorithm are shown in Algorithm 2.1 below where the vertices are ordered according to their fitness function. The reflection process starts by computing the reflected 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 nth 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 nth, the expanded point is accepted. Otherwise, the reflected point is accepted and the iteration contraction, depending on the comparison between the values of the reflected point x_r and the nth point x_n. If the contracted point x_{oc} or x_{ic} is greater than the reflected point x_r, the shrink process is started, where the points are evaluated and the new vertices of simplex at the next iteration are $x'_2, \cdots, x'_{n+1}$ with $x' = x_1 + \phi(x_i - x_1)$, $i = 2, \cdots, n+1$.

3. Overview of the Gravitational Search Algorithm (GSA)

An overview of the main concepts and structure of the gravitational search (GSA) algorithm follows.

- **Main concepts**
  As proposed by Rashedi *et al.* [32], a gravitational search algorithm (GSA) is a population search algorithm modelled on the Newtonian law of gravity and mass interactions. The solutions in the GSA population (called agents) interact with each other through the gravity force, and the performance of each is measured by its mass. Thus each agent is considered to be an object, and some objects move towards another object with heavier mass due to the gravity force, representing a global movements exploration — while the agent with the heavier mass moves more slowly, representing the exploitation step of the algorithm. The best solution is the solution with the heavier mass.
- **Gravitational constant** G
  The gravitational constant G at iteration t is

$$G(t) = G_0 e^{-\alpha t/T}, \tag{3.1}$$

  where G_0 and α are initialised at the beginning of the search and their values are reduced during the search. and T is the total number of iterations.
- **Law of the gravity**
  The objects (masses) obey the law of gravity

$$F = G\,\frac{M_1 M_2}{R^2}, \tag{3.2}$$

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

Algorithm 2.1 The Nelder-Mead Algorithm

1. Let x_i denote the list of vertices in the current simplex, $i = 1, \cdots, n+1$.
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 $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})$.
3. Reflection. Compute the reflected 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,
$\bar{x} = \sum(x_i/n), i = 1, \cdots, n$.
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 expanded 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
$x' = x_1 + \phi(x_i - x_1), i = 2, \cdots, n+1$.
Replace the vertices $x_2, \cdots, x_{n+1}$ with the new vertices $x'_2, \cdots, x'_{n+1}$.
7. Stopping Condition. Order and re-label the vertices of the new simplex as $x_1, x_2, \cdots, x_{n+1}$ such that $f(x_1) \leq f(x_2) \leq \cdots \leq f(x_{n+1})$.
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

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

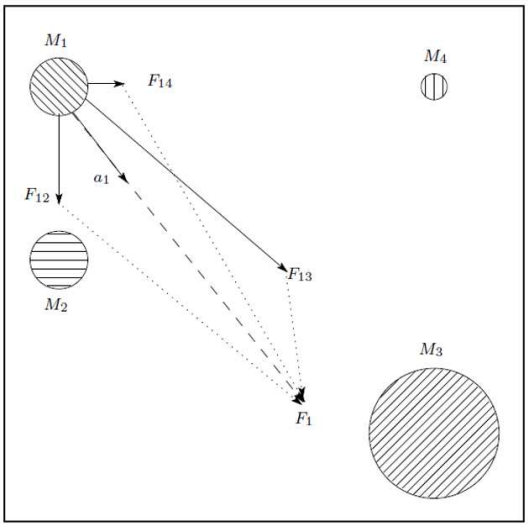


Figure 1: The mass acceleration due to the total force acting on it from the other masses.

where F is a magnitude of the gravitational force, G is gravitational constant, M_1 is the mass of the first object, M_2 is the mass of the second object and R is the distance between the two objects M_1 and M_2; and Newton's second law

$$a = \frac{F}{M}\,, \tag{3.3}$$

giving the acceleration a depending on the applied force F and the object mass M.

- **Acceleration of agents**
  There are three kind of masses, envisaged — viz. active gravitational mass M_a, passive gravitational mass M_p, and inertial mass M_i. The gravitational force F_{ij} acting on mass i by mass j is proportional to the product of M_{aj} and M_{pi} and inversely proportional to the square distance between them, where M_{aj} and M_{pi} are the respective active and passive masses of agents j and i. Also, a_i is is proportional to the gravitational force F_{ij} and inversely proportional to inertia mass M_{ii} of i. Thus from Eqs. (3.2) and (3.3) we have

$$F_{ij} = G\frac{M_{aj} \times M_{pi}}{R^2}\,, \tag{3.4}$$

$$a_i = \frac{F_{ij}}{M_{ii}}\,. \tag{3.5}$$

In order to give a stochastic characterisation to our algorithm, we rewrite the acceleration of object (agent) i as follows.

$$a_i = \frac{\sum_{j=1 j\neq i}^{N} rand_j F_{ij}}{M_{ii}} \tag{3.6}$$

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

where $rand_i$ denotes a random number in $[0,1]$ and N the agent number.

Fig. 1 shows the force F_{1j} acting on mass M_1 due to the mass M_j, and the total force F_1 acting on M_1 that produces the acceleration a.

- **Agent velocity and positions** During the search, the agents update their velocities and positions according to

$$V_i(t+1) = rand_i \times V_i(t) + a_i(t)\,, \tag{3.7}$$

$$X_i(t+1) = X_i(t) \times V_i(t+1)\,, \tag{3.8}$$

where the initial velocity V_0 is generated randomly and $rand_i$ is a uniform random variable in the interval $[0,1]$.

3.1. Gravitational search algorithm

The structure of the standard gravitational search (GSA) algorithm is represented in Algorithm 3.1.

Algorithm 3.1 The standard gravitational search algorithm

1: Set the initial values of gravitational constants G_0 and α.
2: Set the initial iteration $t = 0$.
3: **for** $i = 1; i \le N$ **do**
4: Generate an initial population $X_i(t)$ randomly, where $X_i(t) = (x_i^1(t), x_i^2(t), \cdots, x_i^d(t), \cdots, x_i^n(t))$.
5: **end for**
6: **repeat**
7: Evaluate the fitness function $f(X_i(t))$ for each agent in the population $X(t)$.
8: Assign the best and worst agent in the population $X(t)$.
9: Update the gravitational constant G as in (3.1).
10: **for** $i = 1; i \le N$ **do**
11: **for** $j = i+1; j < N$ **do**
12: Calculate the force acting on agent i from agent j as in (3.10).
13: **end for**
14: Calculate the total force that act on agent i as in (3.11).
15: Calculate inertial mass M_i as in (3.12) and (3.15).
16: Calculate the acceleration of the agent i as in (3.16).
17: Update the velocity of agent i as in (3.7).
18: Update the position of agent i as in (3.8).
19: **end for**
20: Set $t = t + 1$.
21: **until** $t \ge Max_{itr}$
22: Return the best solution.

The main steps can be summarised as follows.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

- **Step 1.** Set the initial values of the gravitational constants G_0 and α, and the iteration counter t.
- **Step 2.** Randomly generate the initial population of N agents, and define the position of each agent as

$$X_i(t) = \left(x_i^1(t), x_i^2(t), \cdots, x_i^d(t), \cdots, x_i^n(t)\right), \quad i = 1, 2, \cdots, N, \tag{3.9}$$

where x_i^d represents the position of the agent i in the d^{th} dimension and n is the space dimension.
- **Step 3.** Repeat the following steps until termination criteria are satisfied:
3.1 evaluate all agents in the population, and assign the best and worst agents;
3.2 update the gravitational constant according to (3.1);
3.3 calculate the force due to agent j acting on agent i at a specific time t as

$$F_{ij}^d(t) = G(t)\frac{M_{pi}(t) \times M_{aj}(t)}{R_{ij}(t) + \epsilon}\left(x_j^d(t) - x_i^d(t)\right), \tag{3.10}$$

where M_{aj} is the active gravitational mass of agent j, M_{pi} is the passive gravitational mass of agent i, $G(t)$ is the gravitational constant at time t, ϵ is a small constant and $R_{ij}(t)$ is the Euclidian distance between the agents i and j;
3.4. at iteration t, calculate the total force acting on agent i as

$$F_i^d(t) = \sum_{j \in Kbest, j \neq i} rand_j F_{ij}^d(t), \tag{3.11}$$

where $Kbest$ is the set of first K agents with the best fitness value and biggest mass, and $rand_j$ is a random number in the interval $[0, 1]$;
3.5. calculate the inertial mass as

$$m_i(t) = \frac{fit_i - worst(t)}{best(t) - worst(t)}, \tag{3.12}$$

where fit_i is the fitness value of the agent i, $best(t)$ and $worst(t)$ are given by

$$best(t) = \min_{i \in \{1, \cdots, N\}} fit_i(t), \tag{3.13}$$

$$worst(t) = \max_{i \in \{1, \cdots, N\}} fit_i(t), \tag{3.14}$$

$$M_i(t) = \frac{m_i(t)}{\sum_{j=1}^N m_j(t)}; \tag{3.15}$$

3.6. calculate the acceleration of agent i as

$$a_i^d(t) = \frac{F_i^d(t)}{M_{ii}(t)}, \tag{3.16}$$

where $F_i(t)$ is the total force acting on agent i given by Eq. (3.11) and M_{ii} is the mass of the i-th agent;

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

3.7. compute the velocity and position of agent i from Eqs. (3.7) and (3.8), respectively; and
3.8. increase the iteration counter until termination criteria are satisfied.

- **Step 4.** Produce the best optimal solution.

4. The Direct Gravitational Search Algorithm (DGSA)

Our direct gravitational search algorithm (DGSA) is presented in Algorithm 4.1.

Algorithm 4.1 Direct gravitational search algorithm

Set the initial values of gravitational constants G_0 and α.
Set the initial iteration $t = 0$.
for $i = 1; i \leq N$ **do**
  Generate an initial population $X_i(t)$ randomly, where $X_i(t) = (x_i^1(t), x_i^2(t), \cdots, x_i^d(t), \cdots, x_i^n(t))$.
end for
repeat
  Evaluate the fitness function $f(X_i(t))$ for each agent in the population $X(t)$.
  Assign the best and worst agent in the population $X(t)$.
  Update the gravitational constant G as in (3.1).
  for $i = 1; i \leq N$ **do**
    for $j = i + 1; j < N$ **do**
      Calculate the force acting on agent i from agent j as in (3.10).
    end for
    Calculate the total force that acts on agent i as in (3.11).
    Calculate inertial mass M_i as in (3.12) and (3.15).
    Calculate the acceleration of the agent i as in (3.16).
    Update the velocity of agent i as in (3.7).
    Update the position of agent i as in (3.8).
  end for
  Set $t = t + 1$.
until $t \geq Max_{itr}$
Apply Nelder-Mead method on the N_{elite} best solutions as shown in Algorithm 2.1.
Return the best solution.

The main steps of this proposed algorithm are the same as the standard GSA, except for the intensification process invoked in the final stage, where the Nelder-Mead method is applied as an intensification process to refine the best obtained solution from the standard GSA after running it for $4d$ iterations. As previously mentioned, this can accelerate the convergence and avoid running the standard GSA for additional iterations with negligible significant improvement.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

5. Numerical Experiments

In order to investigate the efficiency of the proposed DGSA, its general performance was tested on various benchmark functions and compared against the results from several other algorithms. Our DGSA was programmed via MATLAB, and the results from the algorithms are taken from original articles elsewhere. In the following subsections, *inter alia* the DGSA parameter setting and properties of the applied test functions are reported, and its comparative performance is discussed.

5.1. Parameter setting

The parameters of the DGSA and their assigned values are summarised in Table 1. The parameter values are either based on a common setting in the literature or determined through our preliminary numerical experiments.

- **Population size** N**.** The experimental tests show that the best population size is $N = 20$, as increasing this number increases the function evaluations without notable improvement in the function value.
- **Gravitational constant** G_0 **and gravitational constant** α**.** In our experiment, we adopted the values $G_0 = 100$ and $\alpha = 20$ used in Ref. [32].
- **Intensification parameter** N_{elite}**.** In the final intensification stage, we applied a local search using the Nelder-Mead method, starting from the elite solutions (N_{elite}) obtained in the previous search stage, and we set $N_{elite} = 1$. (Increasing the number of selected elite solutions increases the function evaluations.)
- **Maximum number of iterations** $4d$. The main termination criterion in the standard GSA algorithm is the number of iterations, and in our DGSA we ran the standard GSA $4d$ iterations, and the best solution was then passed to the Nelder-Mead method.

5.2. Termination criteria

The standard GSA number of iterations was run for the best solution from the previous stage, which was passed to the Nelder-Mead method for the intensification process. As previously indicated, the Nelder-Mead method can start from a good solution and directly the search to the optimised solution, instead of running the algorithm with negligible further improvement in the results.

5.3. Integer programming optimisation test problems

The efficiency of the DGSA was first tested on 7 benchmark integer programming problems ($FI_1 - FI_7$), as shown in Table 2. 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.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

Table 1: Parameter setting.

Parameters	Definitions	Values
N	Population Size	20
G_0	gravitational constant	100
α	gravitational constant	20
Max_{itr}	Maximum number of iterations for GSA	$4d$
N_{elite}	No. of best solution for final intensification	1

Table 2: Integer programming optimisation test problems.

Test problem	Problem defination
Problem 1 [33]	$FI_1(x) = \|\|x\|\|_1 = \|x_1\| + \cdots + \|x_n\|$
Problem 2 [33]	$FI_2(x) = x^T x = \begin{bmatrix} x_1 & \cdots & x_n \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$
Problem 3 [12]	$FI_3(x) = \begin{bmatrix} 15 & 27 & 36 & 18 & 12 \end{bmatrix} x + x^T \begin{bmatrix} 35 & -20 & -10 & 32 & -10 \\ -20 & 40 & -6 & -31 & 32 \\ -10 & -6 & 11 & -6 & -10 \\ 32 & -31 & -6 & 38 & -20 \\ -10 & 32 & -10 & -20 & 31 \end{bmatrix} x$
Problem 4 [12]	$FI_4(x) = (9x_1^2 + 2x_2^2 - 11)^2 + (3x_1 + 4x_2^2 - 7)^2$
Problem 5 [12]	$FI_5(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$
Problem 6 [31]	$FI_6(x) = 2x_1^2 + 3x_2^2 + 4x_1x_2 - 6x_1 - 3x_2$
Problem 7 [12]	$FI_7(x) = -3803.84 - 138.08x_1 - 232.92x_2 + 123.08x_1^2 + 203.64x_2^2 + 182.25x_1x_2$

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

5.4. Efficiency of the DGSA in solving integer programming problems

In order to investigate the efficiency of combining the standard GSA and the Nelder-Mead method in our DGSA, we compared not only the standard GSA but also the Nelder-Mead method alone applied to 7 benchmark integer functions. The mean function evaluation values for these three algorithms are reported over 50 runs in Table 4. The termination criterion for the three algorithms was the same — viz. reach the global minimum of the

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

Table 4: Efficiency of the Nelder-Mead method in the final stage of the DGSA, for $FI_1 - FI_7$ integer programming problems.

Function	Standard GSA	NM method	DGSA
FI_1	2020	1536.15	**432.4**
FI_2	1060	658.42	**450.8**
FI_3	5160	1025.36	**857.7**
FI_4	1680	375.25	**258.6**
FI_5	7250	1236.45	**951.2**
FI_6	1520	254.15	**245.8**
FI_7	1100	245.47	**215.2**

solution within an error of 10^{-4} before 20,000 function evaluations. The initial solution in the Nelder-Mead method was randomly generated. The results in Table 4 show that invoking the Nelder-Mead method in the final stage enhances the general performance of the DGSA, and in all cases accelerated the search to reach to the optimal solution or near optimal solution faster than the standard GSA (without the Nelder-Mead method) and the Nelder-Mead method alone. Table 4 shows that the DGSA can obtain the optimal or near optimal results faster than the standard GSA.

5.5. Performance of the DGSA in solving integer programming problems

In Fig. 2, the number of iterations versus function values for the set of functions $\{FI_1, FI_2, FI_6, FI_7\}$ (randomly picked) are shown. The solid line represents the performance of the standard GSA after $4d$ iterations, and the best solution was passed to the Nelder-Mead method to refine it. The performance of the Nelder-Mead method is represented by the dotted line. From Fig. 2, it is evident that the DGSA can accelerate the search and avoid running the standard GSA for iterations that produce negligible further improvement in the results.

5.6. Comparison of the DGSA with other algorithms

To further consider its efficiency, the DGSA was compared with four benchmark algorithms (particle swarm optimisation with various algorithms), as follows — cf. also Ref. [29].

- **RWMPSOg**, the Random Walk Memetic Particle Swarm Optimisation (with global variant) combining particle swarm optimisation with random walk as direction exploitation.
- **RWMPSOl**, another Random Walk Memetic Particle Swarm Optimisation (with local variant) combining particle swarm optimisation with random walk as direction exploitation.
- **PSOg**, a standard Particle Swarm Optimisation with global variant without local search method.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

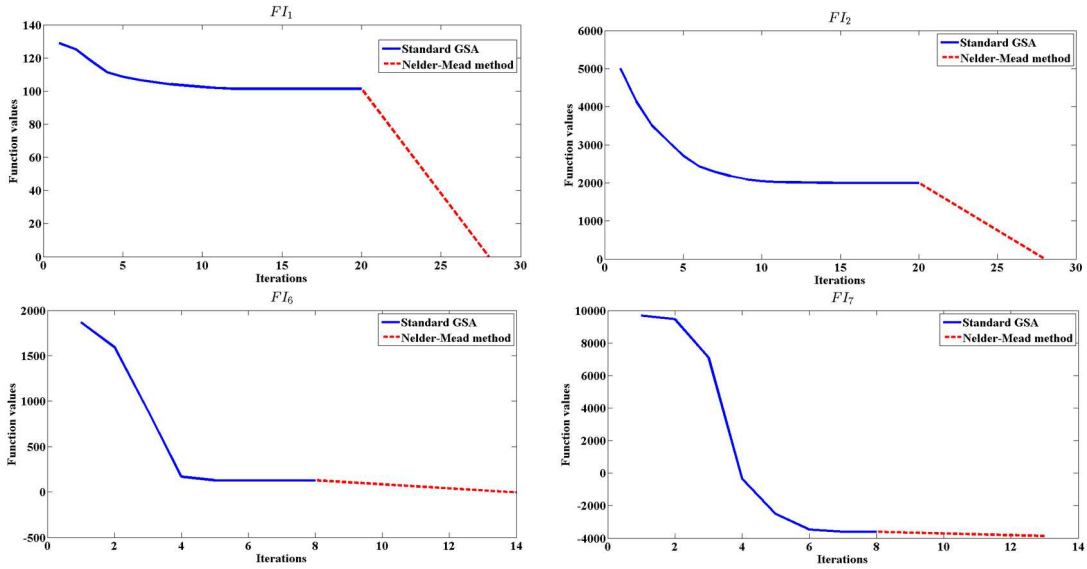


Figure 2: Performance of DGSA in solving integer problems.

- **PSOl**, another Standard Particle Swarm Optimisation with local variant without local search method.

5.6.1. Comparison of RWMPSOg, RWMPSOl, PSOg, PSOl and DGSA in solving integer programming problems.

The five algorithms were tested on 7 benchmark functions, with results for the four previous algorithms (RWMPSOg, RWMPSOl, PSOg, and PSOl) taken from Ref. [29]. In Table 5, the minimum (min), maximum (max), average (Mean), standard deviation (St.D) and success rate (%Suc) for the evaluation function values are reported over 50 runs. A run was considered successful if the algorithm reached the global minimum of the solution within an error of 10^{-4} before the 20,000 function evaluations, and the best results in Table 5 are reported in **boldface** text. The results show that the DGSA was successful for 5 of the 7 functions, and its success rate is 100% for all 7 benchmark functions.

5.7. DGSA and the branch and bound method

Another investigation compared the DGSA against the branch and bound (BB) method [4, 5, 19, 23] in the solution of integer programming problems. The branch and bound method (BB) is one of the most widely for solving optimisation problems, where the main idea is to partition the feasible region of the problem into several sub regions, in an operation called branching. The lower and upper bounds of the function can be determined over these partitions, in an operation called bounding. The BB method is represented in Algorithm 5.1, and the steps involved then summarised below.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

Table 5: Experimental results (min, max, mean, standard deviation and rate of success) of function evaluation for the $FI_1 - FI_7$ 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
	DGSA	425	483	**432.4**	24.005	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
	DGSA	425	475	**450.8**	26.299	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
	DGSA	844	885	**857.7**	18.27	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
	DGSA	240	270	253.4	14.06	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
	DGSA	941	1100	**951.4**	72.02	50
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
	DGSA	240	275	255.6	15.54	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
	DGSA	210	240	**215.2**	13.42	50

- **Step 1.** Partition a relaxed feasible region $M_0 \supset S$, where S is the feasible region of the problem, into a finitely number of subsets M_i.
- **Step 2.** For each subset M_i, determine the lower bound β and the upper bound α

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

Algorithm 5.1 The branch and bound algorithm

Set the feasible region M_0, $M_0 \supset S$.
Set $i = 0$
repeat
  Set $i = i + 1$
  Partition the feasible region M_0 into many subsets M_i.
  For each subset M_i, determine lower bound β, where $\beta = \min\ \beta(M_i)$.
  For each subset M_i, determine upper bound α, where $\alpha = \min\ \alpha(M_i)$.
  if $(\alpha = \beta)||(\alpha - \beta \leq \epsilon)$ **then**
    Stop
  else
    Select some of the subset M_i and partition them.
  end if
  Determine new bound on the new partition elements.
until $(i \leq m)$

such that $\beta(M_i) \leq \inf f(M_i \bigcap S) \leq \alpha(M_i)$, where f is the objective function.

- **Step 3.** Terminate the algorithm if the bounds are equal or very close — i.e. $\alpha = \beta$, or $\alpha - \beta \leq \epsilon$ where ϵ is a predefined positive constant.
- **Step 4.** Otherwise, if the bounds are not equal or very close, select and partition some of the subsets M_i in order to obtain a more refined partition of M_0.
- **Step 5.** Repeat the procedure until termination criteria are satisfied.

5.7.1. Comparison of the BB method and DGSA for integer programming problems

Results from the BB method and the DGSA are shown in Table 6, with the results for the BB method taken from Ref. [18] follow from transforming the initial integer problem programming problem into a continuous problem. The sequential quadratic programming method is used to solve the generated sub-problems for the bounding, while for the branching the BB uses depth first traversal with backtracking. The average (Mean), standard deviation (St.D) and rate of success (Suc) are reported over 30 runs in Table 6, with the best mean evaluation values from the two algorithms fhown in **boldface** text. The results from the DGSA are evidently better than the results from the BB method for 5 of the 7 test functions, and the rate of success for the DGSA is 100% for all functions. The overall results in Table 6 show that the proposed algorithm is faster and more efficient than the BB method for most cases.

5.8. Minimax optimisation test problems

The second type of the tested problems is the minimax problems. The proposed algorithm is tested on 10 benchmark minimax functions as shown in Tables 7. The properties of these functions are reported in Table 8 and the form of each function is listed as follows.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

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

Function	Algorithm	Mean	St.D	Suc
FI_1	BB	1167.83	659.8	30
	DGSA	**429.6**	21.5	30
FI_2	BB	**139.7**	102.6	30
	DGSA	448.15	22.28	30
FI_3	BB	4185.5	32.8	30
	DGSA	**837.6**	15.46	30
FI_4	BB	316.9	125.4	30
	DGSA	**249.47**	12.46	30
FI_5	BB	2754	1030.1	30
	DGSA	**925.26**	68.42	30
FI_6	BB	**211**	15	30
	DGSA	224.15	12.45	30
FI_7	BB	358.6	14.7	30
	DGSA	**212.49**	11.76	30

5.9. Efficiency of the proposed algorithm in solving minimax problems

In order to investigate the importance of the Nelder-Mead method in the final stage as a final intensification process in the proposed algorithm, the standard GSA and the Nelder-Mead method alone were compared with the DGSA in solving 10 minimax benchmark problems. The results in Table 9 were obtained using the same termination criterion, with the search terminated when the optimal solution is reached within an error of 10^{-4} or there were $20,000$ function evaluations, and the best results are again shown in **boldface** text. The results show that invoking the Nelder-Mead method in the final stage enhance the general performance of the DGSA, with the search to reach to the optimal or near optimal solution faster than the standard GSA and the Nelder-Mead method.

5.10. Performance of the DGSA in solving minimax problems

The relative performance of the DGSA in solving minimax problems was investigated by plotting the values of function values versus the number of iterations as shown in Fig. 3 for the four functions $\{FM_1, FM_2, FM_5, FM_{10}\}$ randomly picked. The solid line represents the results of the standard GSA, while the dotted line represents the results of the Nelder-Mead method after passing the best obtained solution from the GSA to the Nelder-Mead method. The results show that the function values from the DGSA rapidly decrease as the number of iterations increases, which verifies that invoking the Nelder-Mead method as an intensification process can accelerate the search for the optimal or near optimal solution.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

Table 7: Minimax optimisation test problems.

Test problem & Problem definition	
Problem 1 [40]	$FM_1(x) = \max\ f_i(x),\ \ i = 1,2,3,$ $f_1(x) = x_1^2 + x_2^4,$ $f_2(x) = (2-x1)^2 + (2-x_2)^2,$ $f_3(x) = 2\exp(-x_1 + x_2)$
Problem 2 [40]	$FM_2(x) = \max\ f_i(x),\ \ i = 1,2,3,$ $f_1(x) = x_1^4 + x_2^2$ $f_2(x) = (2-x1)^2 + (2-x_2)^2,$ $f_3(x) = 2\exp(-x_1 + x_2)$
Problem 3 [40]	$FM_3(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4,$ $g_2(x) = -x_1^2 - x_2^2 - x_3^3 - x_4^2 - x_1 + x_2 - x_3 + x_4 + 8,$ $g_3(x) = -x_1^2 - 2x_2^2 - x_3^2 - 2x_4 + x_1 + x_4 + 10,$ $g_4(x) = -x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5$
Problem 4 [40]	$FM_4(x) = \max f_i(x)\ \ i = 1,\cdots,5$ $f_1(x) = (x_1-10)^2 + 5(x_2-12)^2 + x_3^4 + 3(x_4-11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7,$ $f_2(x) = f_1(x) + 10(2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 - 127),$ $f_3(x) = f_1(x) + 10(7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 - 282),$ $f_4(x) = f_1(x) + 10(23x_1 + x_2^2 + 6x_6^2 - 8x_7 - 196),$ $f_5(x) = f_1(x) + 10(4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7$
Problem 5 [34]	$FM_5(x) = \max\ f_i(x),\ \ i = 1,2,$ $f_1(x) = \lvert x_1 + 2x_2 - 7\rvert,$ $f_2(x) = \lvert 2x_1 + x_2 - 5\rvert$
Problem 6 [34]	$FM_6(x) = \max\ f_i(x),$ $f_i(x) = \lvert x_i\rvert,\ \ i = 1,\cdots,10$
Problem 7 [22]	$FM_7(x) = \max\ f_i(x),\ \ i = 1,2,$ $f_1(x) = (x_1 - \sqrt{(x_1^2+x_2^2)}\cos\sqrt{x_1^2+x_2^2})^2 + 0.005(x_1^2+x_2^2)^2,$ $f_2(x) = (x_2 - \sqrt{(x_1^2+x_2^2)}\sin\sqrt{x_1^2+x_2^2})^2 + 0.005(x_1^2+x_2^2)^2$
Problem 8 [22]	$FM_8(x) = \max\ f_i(x),\ \ i = 1,\cdots,4,$ $f_1(x) = \left(x_1 - (x_4+1)^4\right)^2 + \left(x_2 - \left(x_1 - (x_4+1)^4\right)^4\right)^2 + 2x_3^2 + x_4^2 - 5\left(x_1 - (x_4+1)^4\right)$ $-5\left(x_2 - \left(x1 - (x_4+1)^4\right)^4\right) - 21x_3 + 7x_4,$ $f_2(x) = f_1(x) + 10\Big[\left(x_1 - (x_4+1)^4\right)^2 + \left(x_2 - \left(x_1 - (x_4+1)^4\right)^4\right)^2 + x_3^2 + x_4^2 + \left(x_1 - (x_4+1)^4\right)$ $-\left(x_2 - \left(x_1 - (x_4+1)^4\right)^4\right) + x_3 - x_4 - 8\Big],$ $f_3(x) = f_1(x) + 10\Big[\left(x_1 - (x_4+1)^4\right)^2 + 2\left(x_2 - \left(x_1 - (x_4+1)^4\right)^4\right)^2 + x_3^2 + 2x_4^2 - \left(x_1 - (x_4+1)^4\right)$ $-x_4 - 10\Big],$ $f_4(x) = f_1(x) + 10\Big[\left(x_1 - (x_4+1)^4\right)^2 + \left(x_2 - \left(x_1 - (x_4+1)^4\right)^4\right)^2 + x_3^2 + 2\left(x_1 - (x_4+1)^4\right)$ $-\left(x_2 - \left(x_1 - (x_4+1)^4\right)^4\right) - x_4 - 5\Big]$
Problem 9 [22]	$FM_9(x) = \max\ f_i(x),\ \ i = 1,\cdots,5,$ $f_1(x) = (x_1-10)^2 + 5(x_2-12)^2 + x_3^4 + 3(x_4-11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7,$ $f_2(x) = -2x_1^2 - 2x_3^4 - x_3 - 4x_4^2 - 5x_5 + 127,$ $f_3(x) = -7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 + 282,$ $f_4(x) = -23x_1 - x_2^2 - 6x_6^2 + 8x_7 + 196,$ $f_5(x) = -4x_1^2 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7$
Problem 10 [22]	$FM_{10}(x) = \max\lvert f_i(x)\rvert,\ \ i = 1,\cdots,21,$ $f_i(x) = x_1\exp(x_3t_i) + x_2\exp(x_4t_i) - \frac{1}{1+t_i},$ $t_i = -0.5 + \frac{i-1}{20}$

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

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

Table 9: Efficiency of the Nelder-Mead method in the final stage of the DGSA for $FM_1 - FM_{10}$ minimax problems.

Function	Standard GSA	NM method	DGSA
FM_1	1620.4	325.25	**285.6**
FM_2	1980.6	296.46	**244.4**
FM_3	1800.7	559.26	**1123.2**
FM_4	5220.6	19,250.25	**2479.4**
FM_5	1680.4	295.25	**232.6**
FM_6	1800.5	18,245.48	**811.4**
FM_7	1860.6	955.36	**895.4**
FM_8	7200.4	1745.45	**1053.6**
FM_9	8500.6	19,857.69	**2660.2**
FM_{10}	1440.7	1045.55	**961.4**

5.11. The DGSA and other algorithms

In order to further investigate its efficiency in solving minimax problems, the DGSA was compared with the following three benchmark algorithms.

- HPS2 [13], the Heuristic Pattern Search algorithm 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 [27], the Unified Particle Swarm Optimisation algorithm that combines the global and local variants of the standard PSO and incorporates a stochastic parameter to imitate mutation in evolutionary algorithms.
- RWMPSOg [29], the Random Walk Memetic Particle Swarm Optimisation algorithm (with global variant), which combines particle swarm optimisation with random walk as direction exploitation.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

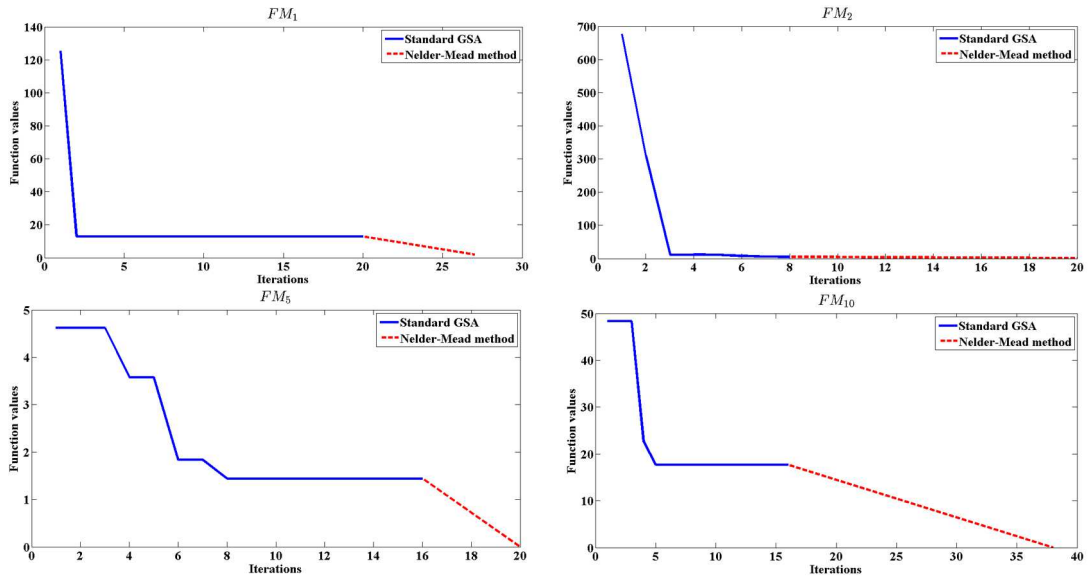


Figure 3: The general performance of DGSA on Minimax problems.

5.11.1. Comparison of the HPS2, UPSOm and RWMPSOg with the DGSA in solving minimax problems

The four comparative algorithms were tested on 10 benchmark functions, with yhe results for the HPS2, UPSOm and RWMPSOg algorithms are taken from Ref. [13]. In Table 10, the average (Avg), standard deviation (SD) and Success rate (%Suc) are reported over 100 runs. The mark (-) for FM_8 from the HPS2 algorithm and FM_2, FM_8 and FM_9 from the RWMPSOg algorithm means that the corresponding results were not reported in Ref. [13]. A run was again considered successful 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 DGSA is successful in most runs and can obtain the objective value of each function faster than the other algorithms, except for functions FM_3, FM_9 and FM_{10} where the HPS2 results are better.

5.12. DGSA and SQP method

Our last test for the DGSA was to compare it with the Sequential Quadratic Programming method (SQP), which involves the main steps summarised below.

5.12.1. Sequential quadratic programming (SQP)

In his PhD thesis [39], the first reference for SQP algorithms, Wilson proposed the Newton-SQP algorithm to solve unconstrained optimisation problems. The development of the secant or variable-metric algorithms led to the extension of these methods to solve the constrained problem.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

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
DGSA	FM_1	**285.6**	14.72	100
	FM_2	**244.4**	16.60	100
	FM_3	1123.2	39.12	100
	FM_4	**2479.4**	147.39	100
	FM_5	**232.4**	12.89	100
	FM_6	**811.4**	24.96	100
	FM_7	**895.4**	35.09	100
	FM_8	**1053.6**	50.90	100
	FM_9	2660.2	75.06	100
	FM_{10}	961.4	15.24	100

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

- **Step 1.** Adopt an initial solution x_0 and initialisation of the Hessian matrix of the objective function.
- **Step 2.** At each iteration, invoke the Broyden–Fletcher–Goldfarb–Shanno (BFGS) method to calculate a positive definite quasi-Newton approximation of the Hessian matrix, updated as follows:

$$H_{n+1} = H_n + \frac{q_n q_n^T}{q_n^T s_n} - \frac{H_n^T H_n}{s_n^T H_n s_n}\,, \tag{5.1}$$

  where $s_n = x_{n+1} - x_n$ and $q_n = \nabla f(x_{n+1})$.
- **Step 3.** Solve the QP problem in z as follows:

$$min\ q(z) = 1/2 z^T H z + c^T z\,. \tag{5.2}$$

- **Step 4.** Calculate the new potential solution by using the solution z_n as follows:

$$x_{n+1} = x_n + \alpha_n z_n\,, \tag{5.3}$$

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

For extensive theoretical discussions of aspects of the SQP algorithm, see Refs. [9, 11].

The DGSA and SQP method were compared for 10 benchmark functions, with the results for the SQP algorithm taken from Ref. [18]. In Table 11, the average (Avg), standard deviation (SD) and Success rate (%Suc) are reported over 30 runs, where once again a run was considered successful if the algorithm reached the global minimum of the solution within an error of 10^{-4} before 20,000 function evaluations. The results show that the DGSA outperforms the SQP algorithm in 7 of the 10 functions, but the results from the SQP algorithm are better for the three functions FM_3, FM_5, and FM_6.

6. Conclusion

A new hybrid algorithm combining a gravitational search algorithm and the Nelder-Mead method is presented, called the Direct Gravitational Search Algorithm (DGSA). Thus the standard gravitational algorithm (GSA) providing wide exploration and deep exploitation is combined with the Nelder-Mead algorithm, a direct search algorithm providing an efficient local search method. The GSA suffers from slow convergence, as do most metaheuristic algorithms. The Nelder-Mead method invoked with the standard GSA accelerates the search and avoids running the algorithm with additional iterations that provide negligible further improvement. The DGSA has been investigated by applying it to 7 benchmark integer problems and 10 minimax problems in comparing it to 9 previous algorithms. The experimental results show that the DGSA usually provides superior results in most cases.

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

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

Function	Algorithm	Mean	St.D	Suc
FM_1	SQP	4044.5	8116.6	24
	DGSA	**281.4**	11.26	30
FM_2	SQP	8035.7	9939.9	18
	DGSA	**239.2**	12.78	30
FM_3	SQP	**135.5**	**21.1**	30
	DGSA	1115.5	29.46	30
FM_4	SQP	20,000	0.0	0.0
	DGSA	**2468.6**	135.48	30
FM_5	SQP	**140.6**	38.5	30
	DGSA	229.6	9.15	30
FM_6	SQP	**611.6**	200.6	30
	DGSA	805.4	21.15	30
FM_7	SQP	15,684.0	7302.0	10
	DGSA	**891.2**	31.48	30
FM_8	SQP	20,000	0.0	0.0
	DGSA	**1039.2**	48.18	30
FM_9	SQP	20,000	0.0	0.0
	DGSA	**2645.29**	69.45	30
FM_{10}	SQP	4886.5	8488.4	22
	DGSA	**1415.8**	14.28	30

Acknowledgments

The authors are grateful to anonymous referees for valuable comments and helpful suggestions that greatly improved the quality of this article. The first author is the recipient of a postdoctoral scholarship provided by the Natural Sciences and Engineering Research Council of Canada (NSERC), which also supports the research of the second author.

References

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

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

[3] N. Bacanin, I. Brajevic and M. Tuba, *Firefly Algorithm applied to integer programming problems*, in Proc. 7th Int. Conf. on Applied Mathematics, Simulation, Modelling (ASM'13), Cambridge, Mass., USA, pp. 143-148, WSEAS Press (2013).

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

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

[5] B. Borchers and J.E. Mitchell, *An improved branch and bound algorithm for mixed integer nonlinear programs*, Computers Op. Res. **21**, 359-367 (1994).
[6] S.A. Chu, P.W. Tsai and J.S. Pan, *Cat swarm optimization*, Lecture Notes Comp. Sc. (subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 4099 LNAI, pp. 854-858 (2006).
[7] M. Dorigo, *Optimization, Learning and Natural Algorithms*, Ph.D. Thesis, Politecnico di Milano, Italy (1992).
[8] D.Z. Du and P.M. Pardalose, *Minimax and Applications*, Kluwer (1995).
[9] R. Fletcher, *Practical Method of Optimization*, Vols. 1 & 2, Wiley (1980).
[10] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco (1979).
[11] P.E. Gill, W. Murray and M.H. Wright, *Practical Optimzization*, Academic Press, London (1981).
[12] A. Glankwahmdee, J.S. Liebman and G.L. Hogg, *Unconstrained discrete nonlinear programming*, Eng. Optimization **4**, 95-107 (1979).
[13] A.C.P. Isabel, E. Santo and E. Fernandes, *Heuristics pattern search for bound constrained minimax problems*, Comp. Sc. Appl. **6784**, 174-184, ICCSA (2011).
[14] R. Jovanovic and M. Tuba, *An ant colony optimization algorithm with improved pheromone correction strategy for the minimum weight vertex cover problem*, Appl. Soft Computing **11**, 5360-5366 (2011).
[15] R. Jovanovic and M. Tuba, *Ant colony optimization Aagorithm with pheromone correction strategy for minimum connected dominating set problem*, Comp. Sc. Inf. Systems **10**, 133-149 (2013).
[16] D. Karaboga and B. Basturk, *A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm*, J. Global Optimization **39**, 459-471 (2007).
[17] J. Kennedy and R.C. Eberhart, *Particle swarm optimization*, in Proc. IEEE Int. Conf. Neural Networks **4**, pp. 1942-1948 (1995).
[18] E.C. Laskari, K.E. Parsopoulos and M.N. Vrahatis, *Particle swarm optimization for integer programming*, in Proc. IEEE 2002 Congress on Evolutionary Computation, Honolulu, pp. 1582-1587 (2002).
[19] E.L. Lawler and D.W. Wood, *Branch and bound methods: A survey*, Operations Res. **14**, 699-719 (1966).
[20] X.L. Li, Z.J. Shao and J.X. Qian. *Optimizing method based on autonomous animals: Fish-swarm algorithm*, Xitong Gongcheng Lilun yu Shijian/System Eng. Theory and Practice **22**, 32-38 (2002).
[21] G. Liuzzi, S. Lucidi and M. Sciandrone, *A derivative-free algorithm for linearly constrained finite minimax problems*, SIAM J. Optimization **16**, 1054-1075 (2006).
[22] L. Lukšan and J. Vlcek, *Test problems for nonsmooth unconstrained and linearly constrained optimization*, Technical Rep. **798**, Institute Comp. Sc., Academy of Sciences of the Czech Republic, Prague (2000).
[23] V.M. Manquinho, J.P. Marques Silva, A.L. Oliveira and K.A. Sakallah, *Branch and bound algorithms for highly constrained integer programs*, Technical Rep., Cadence European Laboratories, Portugal (1997).
[24] J.A. Nelder and R. Mead, *A simplex methods for function minimization*, Computer J. **7**, 308-313 (1965).
[25] G.L. Nemhauser, A.H.G. Rinnooy Kan and M.J. Todd, *Handbooks in OR & MS*, Vol. 1, Elsevier (1989).
[26] G. Nemhauser and L. Wolsey, *Integer and Combinatorial Optimization*, Wiley (1988).
[27] K.E. Parsopoulos and M.N. Vrahatis, *Unified particle swarm optimization for tackling operations*

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press

research problems, in Proc. IEEE 2005 Swarm Intelligence Symposium, Pasadena, USA, pp. 53-59 (2005).
[28] M.K. Passino, *Biomimicry of bacterial foraging for distributed optimization and control*, Control Systems, IEEE **22**, 52-67 (2002).
[29] Y.G. Petalas, K.E. Parsopoulos and M.N. Vrahatis, *Memetic particle swarm optimization*, Ann. Op. Res. **156**, 99-127 (2007).
[30] E. Polak, J.O. Royset and R.S. Womersley, *Algorithms with adaptive smoothing for finite minimax problems*, J. Optimization Theory Applic. **119**, 459-484 (2003).
[31] S.S. Rao, *Engineering Optimization-Theory and Practice*, Wiley (1994).
[32] E. Rashedi, H. Nezamabadi-pour, and S. Saryazdi, *GSA: A gravitational search algorithm*, Inf. Sc. **179**, 2232-2248 (2009).
[33] G. Rudolph, *An evolutionary algorithm for integer programming*, in Parallel Problem Solving from Nature 3, Y. Davidor, H.P. Schwefel and R. Männer (Eds), pp. 139-148 (1994).
[34] H.P. Schwefel, *Evolution and Optimum Seeking*, Wiley (1995).
[35] R. Tang, S. Fong, X.S. Yang and S. Deb, *Wolf search algorithm with ephemeral memory*, in Digital Information Management, 2012 Seventh Int. Conf. Digital Inf. Management (ICDIM), pp. 165-172 (2012).
[36] D. Teodorovic and M. DellOrco, Bee colony optimizationa 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 (13-16 September 2005), Poznan: Publishing House of the Polish Operational and System Research, pp. 51-60 (2005).
[37] M. Tuba, N. Bacanin and N. Stanarevic, *Adjusted artificial bee colony (ABC) algorithm for engineering problems*, WSEAS Trans. Computers **11**, 111-120 (2012).
[38] M. Tuba, M. Subotic and N. Stanarevic, *Performance of a modified cuckoo search algorithm for unconstrained optimization problems*, WSEAS Trans. Systems **11**, 62-74 (2012).
[39] B. Wilson, *A Simplicial Algorithm for Concave Programming*, Ph.D. Thesis, Harvard University (1963).
[40] S. Xu, *Smoothing method for minimax problems*, Comp. Optimization Appl. **20**, 267-279 (2001).
[41] X.S. Yang, *Firefly algorithm, stochastic test functions and design optimization*, Int. J. Bio-Inspired Comp. **2**, 78-84 (2010).
[42] X.S. Yang, *A new meta-heuristic bat-inspired algorithm*, in Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), pp. 65-74 (2010).
[43] S. Zuhe, A. Neumaier and M.C. Eiermann, *Solving minimax problems by interval methods*, BIT **30**, 742-751 (1990).

https://doi.org/10.4208/eajam.030915.210416a Published online by Cambridge University Press