A Plant Propagation Algorithm for Constrained Engineering Optimisation Problems

Muhammad Sulaiman, Abdellah Salhi, Birsen Irem Selamoglu, and Omar Bahaaldin Kirikchi

1 Department of Mathematical Sciences, University of Essex, Colchester CO4 3SQ, UK
2 Department of Mathematics, Abdul Wali Khan University, Mardan KPK, Pakistan

Correspondence should be addressed to Muhammad Sulaiman; sulaiman513@yahoo.co.uk

Received 17 February 2014; Accepted 6 April 2014; Published 7 May 2014

Optimisation problems arising in industry are some of the hardest, often because of the tight specifications of the products involved. They are almost invariably constrained and they involve highly nonlinear, and non-convex functions both in the objective and in the constraints. It is also often the case that the solutions required must be of high quality and obtained in realistic times. Although there are already a number of well performing optimisation algorithms for such problems, here we consider the novel Plant Propagation Algorithm (PPA) which on continuous problems seems to be very competitive. It is presented in a modified form to handle a selection of problems of interest. Comparative results obtained with PPA and state-of-the-art optimisation algorithms of the Nature-inspired type are presented and discussed. On this selection of problems, PPA is found to be as good as and in some cases superior to these algorithms.

1. Introduction

Optimisation problems in design engineering are often highly nonlinear, constrained and involving continuous as well as discrete variables [1–5]. It is also often the case that some of the constraints are active at the global optimum [6]. This means that feasible approximate solutions are that much harder to find. There is a variety of algorithms for these problems, some exact, and others approximate. In the exact category, one can name Branch-and-Bound, [7], Recursive Quadratic Programming, [8], the Cutting Plane Algorithm [9], Bender’s decomposition [10]. Of the approximate variety, one can name Simulated Annealing, [11–13], the Genetic Algorithm [14–16], and the Particle Swarm Optimisation algorithm, [17, 18], to name a few. The latter category is often referred to as the metaheuristic algorithms. In general, they are characterised by two aspects of the search: exploration of the overall search space and exploitation of good areas in order to find local optima, [19–22].

A new metaheuristic, the Plant Propagation Algorithm (PPA), has recently been introduced [23]. PPA is nature inspired [19, 23, 24]; it emulates the way plants, in particular the strawberry plant, propagate. A basic PPA has been described and tested on single objective as well as multiobjective continuous optimization problems in [23]. The test problems, though standard, were of low dimension. The results showed that PPA has merits and deserves further investigation on higher dimensional problem instances as well as problems arising in practice, for these are often very challenging. PPA is attractive because, among other things, it is simple to describe and implement; it also involves only few parameters that need arbitrary setting unlike most other metaheuristics. Here, it will be tested on constrained
optimisation problems arising in engineering. The paper is organised as follows. Section 2 describes PPA. Section 3 presents a modified version to handle constrained problems. Section 4 records the results obtained with PPA and a number of other heuristics. In Section 5 a conclusion and ideas for further investigation are given. The paper includes appendices that describe the problems considered.

2. The Strawberry Algorithm as PPA

The Strawberry algorithm is an exemplar PPA which can be seen as a multipath following algorithm unlike Simulated Annealing (SA) [13, 25], for instance, which is a single path following algorithm. It can, therefore, be perceived as a generalisation of SA and other path-following algorithms [26].

Exploration and exploitation are properties that effective global optimisation algorithms have [19, 24, 26]. Exploration refers to the property of covering the search space, while exploitation refers to the property of searching nearer to good solutions for local optima.

Consider what a strawberry plant and possibly any plant which propagates through runners will do to optimize its survival. If it is in a good spot of the ground, with enough water, nutrients, and light, then it is reasonable to assume that there is no pressure on it to leave that spot to guarantee its survival. So, it will send only a few runners that will give new strawberry plants and occupy the neighbourhood as best they can. If, on the other hand, the mother plant is in a spot that is poor in water, nutrients, light, or any one of these necessary for a plant to survive, then it will try to find a better spot for its offspring. Therefore, it will send few runners further afield to explore distant neighbourhoods. One can also assume that it will send only a few, since sending a long runner is a big investment for a plant which is in a poor spot. We may further assume that the quality of the spot (abundance of nutrients, water, and light) is reflected in the growth of the plant. With this in mind and the following notation, PPA can be described as follows.

A plant $p_i$ is in spot $X_i$ in dimension $n$. This means $X_i = \{x_{ij}, \text{ for } j = 1, \ldots, n\}$. Let $NP$ be the number of strawberry plants to be used initially, and the PPA algorithm described in pseudo-code Algorithm 1, relies on the following strategy [23].

(i) Strawberry plants which are in good spots propagate by generating many short runners.

(ii) Those in poor spots propagate by generating few long runners.

It is clear that, in the above description, exploitation is implemented by sending many short runners by plants in good spots, while exploration is implemented by sending few long runners by plants in poor spots.

The parameters used in PPA are the population size $NP$ which is the number of strawberry plants, the maximum number of generations $g_{\text{max}}$, and the maximum number of possible runners $n_{\text{max}}$ per plant. $g_{\text{max}}$ is effectively the stopping criterion in this initial version of PPA. The algorithm uses the objective function value at different plant positions $X_i$, $i = 1, \ldots, NP$, in a normalised form $N_j$ to rank them as would a fitness function in a standard genetic algorithm (note that, unlike in the GA, individuals in PPA are clones of the mother plant; they do not improve from generation to generation). The number of plant runners $n_{ij}^\alpha$, calculated according to (1) below, has length $dx^\alpha$ calculated using the normalised form of the objective value at $X_j$, each giving a $dx^\alpha \in (-1,1)^n$, as calculated with (2) below. After all individuals/plants in the population have sent out their allocated runners, new plants are evaluated and the whole increased population is sorted. To keep the population constant, individuals with lower growth are eliminated. The number of runners allocated to a given plant is proportional to its fitness as in

$$n_{ij}^\alpha = [n_{\text{max}}N_j^\alpha], \quad \alpha \in (0,1). \quad (1)$$

Every solution $X_i$ generates at least one runner and its length to the growth as in (2) below:

$$dx^\alpha_j = 2(1 - N_j^{\alpha - 0.5}), \quad \text{for } j = 1, \ldots, n, \quad (2)$$

where $n$ is the problem dimension. Having calculated $dx^\alpha_j$, the extent to which the runner will reach, the search equation that finds the next neighbourhood to explore is

$$y_{ij} = x_{ij} + (b_j - a_j)dx^\alpha_j, \quad \text{for } j = 1, \ldots, n. \quad (3)$$

If the bounds of the search domain are violated, the point is adjusted to be within the domain $[a_j, b_j]$, where $a_j$ and $b_j$ are lower and upper bounds delimiting the search space for the $j$th coordinate.

3. An Effective Implementation of PPA for Constrained Optimization

In this implementation of PPA, the initial population is crucial; we run the algorithm a number of times from randomly generated populations. The best individual from each run forms a member of the initial population. The number of runs to generate the initial population is $NP$; therefore, the population size is $r = NP$. In the case of mixed integer problems, the integer variable values are fixed when they are showing a trend to converge to some values. This trend is monitored by calculating the number of times their values have not changed. When this number is greater than a certain threshold, the variables are fixed for the rest of the run. This strategy seems to work on the problems considered. Let $pop$ be a general matrix containing the population of a given run. Its rows correspond to individuals. The following equation is used to generate a random population for each of the initial runs:

$$x_{ij} = a_j + (b_j - a_j)\alpha, \quad j = 1, \ldots, n, \quad (4)$$

where $x_{ij} \in [a_j, b_j]$ is the $j$th entry of solution $X_i$ and $a_j$ and $b_j$ are the $j$th entries of the lower and upper bounds describing the search space of the problem and $\alpha \in (0,1)$. 

In this implementation of PPA, the initial population is crucial; we run the algorithm a number of times from randomly generated populations. The best individual from each run forms a member of the initial population. The number of runs to generate the initial population is NP; therefore, the population size is $r = NP$. In the case of mixed integer problems, the integer variable values are fixed when they are showing a trend to converge to some values. This trend is monitored by calculating the number of times their values have not changed. When this number is greater than a certain threshold, the variables are fixed for the rest of the run. This strategy seems to work on the problems considered. Let $pop$ be a general matrix containing the population of a given run. Its rows correspond to individuals. The following equation is used to generate a random population for each of the initial runs:

$$x_{ij} = a_j + (b_j - a_j)\alpha, \quad j = 1, \ldots, n, \quad (4)$$

where $x_{ij} \in [a_j, b_j]$ is the $j$th entry of solution $X_i$ and $a_j$ and $b_j$ are the $j$th entries of the lower and upper bounds describing the search space of the problem and $\alpha \in (0,1)$.
In the main body of the algorithm, before updating the population we create a temporary population Φ to hold new solutions generated from each individual in the population. Then we implement three rules with fixed modification parameter \( P_m \), chosen here, as \( P_m = 0.8 \). The first two rules are implemented if the population is initialized randomly. Rule 01 uses the following equation to update the population:

\[
x_{i,j}^* = x_{i,j} (1 + \beta), \quad j = 1, \ldots, n,
\]

where \( \beta \in [-1, 1] \) and \( x_{i,j}^* \in [a, b] \).

The generated individual \( x_i^* \) is evaluated according to the objective function and is stored in Φ. In rule 02 another individual is created with the same modification parameter \( P_m = 0.8 \) as in the following equation:

\[
x_{i,j}^* = x_{i,j} + (x_{i,j} - x_{k,j}) \beta, \quad j = 1, \ldots, n,
\]

where \( \beta \in [-1, 1] \), \( x_{i,j}^* \in [a, b] \), \( i, k \) are mutually exclusive indices and are different from \( i \).

The generated individual \( x_i^* \) is evaluated according to the objective function and is stored in Φ. The first two rules are applicable for \( r \leq NP \) the number of runs. For \( r > NP \) the algorithm also tries to recognise entries which are settling to their final values through a counter \( IN \). If the \( j \)th entry in current population has a low \( IN \) value, then it is modified by implementing (7); otherwise it is left as it is. The value (for \( IN \)) that is suggested by experimentation over a number of problems is 4. The following equation is used when modification is necessary:

\[
x_{i,j}^* = x_{i,j} + (x_{i,j} - x_{k,j}) \beta, \quad j = 1, \ldots, n,
\]

where \( \beta \in [-1, 1] \), \( x_{i,j}^* \in [a, b] \), and \( k \) is different from \( i \). To keep the size of the population constant, the extra plants at the bottom of the sorted population are eliminated.

### 4. Examples of Structural Engineering Optimization Problems

PPA as explained in the pseudo-code Algorithm 1 is extended to cater for constrained optimisation problems to be found in the appendices [6, 27]. This extended version of PPA is fully explained in the pseudo-code Algorithm 2. Note that the penalty function approach is used to handle the constraints [19, 22]. Equations are first transformed into inequality constraints before they are taken into consideration. Table 1 records the parameter values used in the implementation. Column 4 shows the value of \( P_m \) used throughout the experiments. This value has been found through experimentation on the problem described in Appendix B. The different runs are represented in Figure 1 where for 40 trials corresponding to the 30000 function evaluations threshold, \( P_m = 0.8 \) seems to be the optimal value for this parameter. Other aspects of the extended algorithm such as exploration and exploitation are investigated in Figures 2 and 3. These figures, as one expects, show that the magnitudes of the steps/perturbations of the plant positions, that is the lengths of the runners, get shorter and shorter as the search progresses. Figure 4 is a representation of the convergence of the objective values of the problems described in Appendices A and C. In both cases, these values fluctuate wildly before they settle down to very good approximate values. The numerical results of the experiments on all problems described in Appendices A, through G are compiled in Tables 2, 3, 4, 5, 6, 7, and 8.

### 5. Conclusion

We have implemented PPA to solve seven well known difficult constrained optimization problems arising in engineering design with continuous domains. PPA found either near best known solutions or optimal ones to all of them. The results are compared to those obtained with other algorithms found in the literature, namely GA (and variants of it, here denoted EC and EP), PSO, HSA (and variants of it...
### Table 1: Parameters used in PPA for seven engineering problems.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Population size</th>
<th>Maximum iteration</th>
<th>(P_m)</th>
<th>Max function evaluations</th>
<th>Number of runners</th>
<th>Runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Welded beam</td>
<td>40</td>
<td>20</td>
<td>0.8</td>
<td>30000</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>Pressure vessel</td>
<td>40</td>
<td>20</td>
<td>0.8</td>
<td>30000</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>Spring design</td>
<td>40</td>
<td>25</td>
<td>0.8</td>
<td>30000</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>Speed reducer</td>
<td>40</td>
<td>20</td>
<td>0.8</td>
<td>30000</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>Constrained Problem I</td>
<td>40</td>
<td>20</td>
<td>0.8</td>
<td>24000</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>Constrained Problem II</td>
<td>40</td>
<td>20</td>
<td>0.8</td>
<td>24000</td>
<td>3</td>
<td>100</td>
</tr>
<tr>
<td>Himmelblau's function</td>
<td>40</td>
<td>25</td>
<td>0.8</td>
<td>30000</td>
<td>3</td>
<td>100</td>
</tr>
</tbody>
</table>

### Table 2: Welded beam design optimisation.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(w)</td>
<td>0.2088</td>
<td>0.2442</td>
<td>0.2057</td>
<td>0.2057</td>
<td>0.2057</td>
<td>0.2057</td>
<td>0.2057</td>
<td>0.2057</td>
</tr>
<tr>
<td>(h)</td>
<td>0.2100</td>
<td>0.2443</td>
<td>0.2057</td>
<td>0.2057</td>
<td>0.2057</td>
<td>0.2057</td>
<td>0.2057</td>
<td>0.2057</td>
</tr>
<tr>
<td>(g_1(x))</td>
<td>-0.3378</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-1.2733(E-11)</td>
</tr>
<tr>
<td>(g_2(x))</td>
<td>-353.9026</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>-3.3278(E-11)</td>
</tr>
<tr>
<td>(g_3(x))</td>
<td>-0.0012</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-5.5511(E-17)</td>
<td>-5.5511(E-17)</td>
<td>-5.5511(E-17)</td>
<td>-1.64(E-13)</td>
</tr>
<tr>
<td>(g_4(x))</td>
<td>-3.4118</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-3.4329</td>
<td>-3.4329</td>
<td>-3.4329</td>
<td>-3.4329</td>
</tr>
<tr>
<td>(g_5(x))</td>
<td>-0.0838</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.0807</td>
<td>-0.0807</td>
<td>-0.0807</td>
<td>-0.0807</td>
</tr>
<tr>
<td>(g_6(x))</td>
<td>-0.2356</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.2355</td>
<td>-0.2355</td>
<td>-0.2355</td>
<td>-0.2355</td>
</tr>
<tr>
<td>(g_7(x))</td>
<td>-363.2323</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-9.0949(E-13)</td>
<td>-9.0949(E-13)</td>
<td>-9.0949(E-13)</td>
<td>-6.2755(E-11)</td>
</tr>
<tr>
<td>(f(x))</td>
<td>1.7483</td>
<td>2.38</td>
<td>1.7248</td>
<td>1.7248</td>
<td>1.7248</td>
<td>1.7248</td>
<td>1.7248</td>
<td>1.7248</td>
</tr>
</tbody>
</table>

*Not available.*

### Table 3: Pressure vessel design optimisation.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(d_1)</td>
<td>1.125</td>
<td>1.125</td>
<td>1.125</td>
<td>1.125</td>
<td>1.125</td>
<td>1.125</td>
<td>0.7781</td>
</tr>
<tr>
<td>(d_2)</td>
<td>0.625</td>
<td>0.625</td>
<td>0.625</td>
<td>0.625</td>
<td>0.625</td>
<td>0.625</td>
<td>0.3846</td>
</tr>
<tr>
<td>(r)</td>
<td>48.97</td>
<td>58.1978</td>
<td>58.2789</td>
<td>58.2901</td>
<td>58.2874</td>
<td>58.2901</td>
<td>40.3996</td>
</tr>
<tr>
<td>(L)</td>
<td>106.72</td>
<td>44.2930</td>
<td>43.7549</td>
<td>43.6926</td>
<td>43.7075</td>
<td>43.6927</td>
<td>200.0</td>
</tr>
<tr>
<td>(g_1(x))</td>
<td>-0.1799</td>
<td>0.00017</td>
<td>-0.0002</td>
<td>0.0000</td>
<td>-5.2038(E-005)</td>
<td>-5.2038(E-005)</td>
<td>-3.595(E-12)</td>
</tr>
<tr>
<td>(g_2(x))</td>
<td>-0.1578</td>
<td>-0.0697</td>
<td>-0.0690</td>
<td>-0.0689</td>
<td>-0.0689</td>
<td>-0.0689</td>
<td>1.441(E-12)</td>
</tr>
<tr>
<td>(g_3(x))</td>
<td>-97.760</td>
<td>-974.3</td>
<td>-3.7162</td>
<td>-2.0150</td>
<td>-0.6122</td>
<td>-0.0705</td>
<td>1.164(E-9)</td>
</tr>
<tr>
<td>(g_4(x))</td>
<td>-133.28</td>
<td>-195.707</td>
<td>-196.245</td>
<td>-196.307</td>
<td>-196.29</td>
<td>-430.0</td>
<td>-4.00</td>
</tr>
<tr>
<td>(f(x))</td>
<td>7980.894</td>
<td>7204.479</td>
<td>7198.433</td>
<td>7197.730</td>
<td>7197.896</td>
<td>7197.730</td>
<td>5885.3327</td>
</tr>
</tbody>
</table>

### Table 4: Minimization of the weight of a compression spring.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>0.0500</td>
<td>0.0533</td>
<td>0.0519</td>
<td>0.0514</td>
<td>0.0500</td>
<td>0.0518</td>
<td>0.0515</td>
</tr>
<tr>
<td>(x_2)</td>
<td>0.3159</td>
<td>0.3991</td>
<td>0.3639</td>
<td>0.3513</td>
<td>0.3173</td>
<td>0.3608</td>
<td>0.3541</td>
</tr>
<tr>
<td>(g_1(x))</td>
<td>-0.00001</td>
<td>0.00001</td>
<td>-0.00001</td>
<td>-0.0033</td>
<td>-4.7653(E-6)</td>
<td>-6.2196(E-6)</td>
<td>-5.0(E-15)</td>
</tr>
<tr>
<td>(g_2(x))</td>
<td>-0.0037</td>
<td>-0.00001</td>
<td>-0.0002</td>
<td>-1.0970(E-4)</td>
<td>-1.8124(E-4)</td>
<td>-2.8408(E-7)</td>
<td>-1.3901(E-9)</td>
</tr>
<tr>
<td>(g_3(x))</td>
<td>-3.9383</td>
<td>-4.1238</td>
<td>-4.0613</td>
<td>-4.0263</td>
<td>-3.9672</td>
<td>-4.0618</td>
<td>-4.0487</td>
</tr>
<tr>
<td>(g_4(x))</td>
<td>-0.7560</td>
<td>-0.6982</td>
<td>-0.7226</td>
<td>-0.7312</td>
<td>-0.7550</td>
<td>-0.7248</td>
<td>-0.7294</td>
</tr>
<tr>
<td>(f(x))</td>
<td>0.01283</td>
<td>0.01273</td>
<td>0.01268</td>
<td>0.01266</td>
<td>0.01272</td>
<td>0.01266</td>
<td>0.01266</td>
</tr>
</tbody>
</table>
Table 5: Speed reducer design optimisation.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x&lt;sub&gt;1&lt;/sub&gt;</td>
<td>3.50</td>
<td>3.50</td>
<td>3.50</td>
<td>3.50</td>
<td>3.4922</td>
<td>3.4971</td>
</tr>
<tr>
<td>x&lt;sub&gt;2&lt;/sub&gt;</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
<td>0.70</td>
</tr>
<tr>
<td>x&lt;sub&gt;3&lt;/sub&gt;</td>
<td>17.0</td>
<td>17.0</td>
<td>17.0</td>
<td>17.0</td>
<td>17.0</td>
<td>17.0</td>
</tr>
<tr>
<td>x&lt;sub&gt;4&lt;/sub&gt;</td>
<td>7.30</td>
<td>7.30</td>
<td>7.30</td>
<td>7.30</td>
<td>7.30</td>
<td>7.30</td>
</tr>
<tr>
<td>x&lt;sub&gt;5&lt;/sub&gt;</td>
<td>7.80</td>
<td>7.71</td>
<td>7.7159</td>
<td>7.7153</td>
<td>7.80</td>
<td>7.80</td>
</tr>
<tr>
<td>x&lt;sub&gt;6&lt;/sub&gt;</td>
<td>3.3502</td>
<td>3.35</td>
<td>3.3502</td>
<td>3.3502</td>
<td>3.3496</td>
<td>3.3500</td>
</tr>
<tr>
<td>x&lt;sub&gt;7&lt;/sub&gt;</td>
<td>5.2866</td>
<td>5.29</td>
<td>5.2869</td>
<td>5.2866</td>
<td>5.2846</td>
<td>5.2866</td>
</tr>
<tr>
<td>g&lt;sub&gt;1&lt;/sub&gt;(x)</td>
<td>-0.0739</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.07185</td>
<td>-0.07317</td>
</tr>
<tr>
<td>g&lt;sub&gt;2&lt;/sub&gt;(x)</td>
<td>-0.1979</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.1962</td>
<td>-0.1973</td>
</tr>
<tr>
<td>g&lt;sub&gt;3&lt;/sub&gt;(x)</td>
<td>-0.4991</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.4988</td>
<td>-0.4990</td>
</tr>
<tr>
<td>g&lt;sub&gt;4&lt;/sub&gt;(x)</td>
<td>-0.9014</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.9013</td>
<td>-0.9014</td>
</tr>
<tr>
<td>g&lt;sub&gt;5&lt;/sub&gt;(x)</td>
<td>0.0000</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>4.6229E - 4</td>
<td>1.667E - 4</td>
</tr>
<tr>
<td>g&lt;sub&gt;6&lt;/sub&gt;(x)</td>
<td>-5.000E - 16</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>1.816E - 3</td>
<td>6.564E - 4</td>
</tr>
<tr>
<td>g&lt;sub&gt;7&lt;/sub&gt;(x)</td>
<td>-0.7025</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.70250</td>
<td>-0.7025</td>
</tr>
<tr>
<td>g&lt;sub&gt;8&lt;/sub&gt;(x)</td>
<td>-1.000E - 16</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>2.2248E - 3</td>
<td>8.0440E - 4</td>
</tr>
<tr>
<td>g&lt;sub&gt;9&lt;/sub&gt;(x)</td>
<td>-0.5833</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.5842</td>
<td>-0.58366</td>
</tr>
<tr>
<td>g&lt;sub&gt;10&lt;/sub&gt;(x)</td>
<td>-0.0513</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.0514</td>
<td>-0.05136</td>
</tr>
<tr>
<td>g&lt;sub&gt;11&lt;/sub&gt;(x)</td>
<td>-0.0108</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>-0.01114</td>
<td>-0.0108</td>
</tr>
<tr>
<td>f(x)</td>
<td>2,996.3481</td>
<td>2994.4</td>
<td>2994.9</td>
<td>2994.4</td>
<td>2994.4449</td>
<td>2996.1137</td>
</tr>
</tbody>
</table>

*Not available.

Table 6: Constrained optimization problem I.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x&lt;sub&gt;1&lt;/sub&gt;</td>
<td>0.8080</td>
<td>0.8350</td>
<td>0.8343</td>
<td>0.8230</td>
<td>0.8229</td>
<td>0.8298</td>
<td>0.8228</td>
</tr>
<tr>
<td>x&lt;sub&gt;2&lt;/sub&gt;</td>
<td>0.8854</td>
<td>0.9125</td>
<td>0.9121</td>
<td>0.9113</td>
<td>0.9113</td>
<td>0.9098</td>
<td>0.9114</td>
</tr>
<tr>
<td>g&lt;sub&gt;1&lt;/sub&gt;(x)</td>
<td>3.7E - 2</td>
<td>1E - 2</td>
<td>5E - 3</td>
<td>1.1880E - 4</td>
<td>3.8744E - 5</td>
<td>7.8953E - 05</td>
<td>7.05E - 9</td>
</tr>
<tr>
<td>g&lt;sub&gt;2&lt;/sub&gt;(x)</td>
<td>5.2E - 2</td>
<td>-7E - 2</td>
<td>5.4E - 3</td>
<td>3.5443E - 4</td>
<td>1.8967E - 4</td>
<td>9.1722E - 05</td>
<td>1.73E - 8</td>
</tr>
<tr>
<td>f(x)</td>
<td>1.4339</td>
<td>2.3772</td>
<td>1.3770</td>
<td>1.3931</td>
<td>1.3932</td>
<td>1.3774</td>
<td>1.3935</td>
</tr>
</tbody>
</table>

![Figure 3: Exploration characteristic of PPA using rule 1.](image)

![Figure 4: Convergence plot of welded beam design and tension compression spring design optimization problems over 100 trial runs.](image)
(1) Initialization: Generate a population $P = \{X_i, i = 1, \ldots, NP\}$;
(2) $g \leftarrow 1$;
(3) for $g = 1$ to $g_{\text{max}}$ do
(4) Compute $N_i = f(X_i), \forall X_i \in P$;
(5) Sort $P$ in ascending order of $N$ (for minimization);
(6) Create new population $\Phi$;
(7) for each $X_i, i = 1, \ldots, NP$ do
(8) $\alpha_i \leftarrow$ set of runners where both the size of the set and the distance for each runner (individually) are proportional to $N_i$, the normalized objective value;
(9) $\Phi \leftarrow \Phi \cup \alpha_i$ {append to population};
(10) end for
(11) $P \leftarrow \Phi$ {new population};
(12) end for
(13) Return $P$, the population of solutions.

Algorithm 1: Pseudocode of PPA, [23].

(1) Initialization: $g_{\text{max}} \leftarrow$ Maximum number of generations; $NP \leftarrow$ population size; $r \leftarrow$ trial run
(2) if $r \leq NP$ then
(3) Create a random population of plants $\text{pop} = \{X_i | i = 1, 2, \ldots, NP\}$, using (4) and gather the best solutions.
(4) end if
(5) while $r > NP$ do
(6) Use population $\text{pop}_g$ formed by gathering all best solutions from previous runs.
   Calculate $IN_j$ value for each column $j$ of $\text{pop}_g$ (see Section 3).
(7) end while
(8) Evaluate the population. In case of $\text{pop}_g$ the algorithm does not need to evaluate the population,
(9) Set number of runners, $n_r = 3$,
(10) while ($n_{\text{gen}} < g_{\text{max}}$) or ($n_{\text{eval}} < \text{max}_{\text{eval}}$) do
(11) Create $\Phi$;
(12) for $i = 1$ to $NP$ do
(13) for $k = 1$ to $n_r$ do
(14) if $r \leq NP$ then
(15) if rand $\leq P_m$ then
(16) Generate a new solution $X^*$ according to (5);
(17) Evaluate it and store it in $\Phi$;
(18) end if
(19) if rand $\leq P_m$ then
(20) Generate a new solution $X^*$ according to (6);
(21) Evaluate it and store it in $\Phi$;
(22) end if
(23) else
(24) for $j = 1$ to $n$ do
(25) if ($IN_j < 4$) or (rand $\leq P_m$) then
(26) update the $j$th entry of $X_i, i = 1, 2, \ldots, NP$, according to (7);
(27) end if
(28) Evaluate new solution $X^*$ and store it in $\Phi$;
(29) end for
(30) end if
(31) end for
(32) end for
(33) Add $\Phi$ to current population;
(34) Sort the population in ascending order of the objective values;
(35) Update current best;
(36) end while
(37) Return: Updated population.

Algorithm 2: Pseudocode of PPA for constrained optimisation.
Table 7: Constrained optimization problem II.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>(x_1)</th>
<th>(x_2)</th>
<th>(f(x))</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deb [38]</td>
<td>*</td>
<td>*</td>
<td>13.5895</td>
</tr>
<tr>
<td>GA with PS ((R = 0.01))</td>
<td>*</td>
<td>*</td>
<td>13.5910</td>
</tr>
<tr>
<td>GA with TS-R</td>
<td>*</td>
<td>*</td>
<td>13.5908</td>
</tr>
<tr>
<td>HSA [30]</td>
<td>2.2468</td>
<td>2.3818</td>
<td>13.5908</td>
</tr>
<tr>
<td>HSA [29]</td>
<td>2.2468</td>
<td>2.3821</td>
<td>13.5908</td>
</tr>
<tr>
<td>PHS [27]</td>
<td>2.2480</td>
<td>2.4066</td>
<td>13.6117</td>
</tr>
<tr>
<td>IPHS [27]</td>
<td>2.2468</td>
<td>2.3818</td>
<td>13.5908</td>
</tr>
<tr>
<td>PPA</td>
<td>2.2468</td>
<td>2.3818</td>
<td>13.5908</td>
</tr>
<tr>
<td>Optimal solution</td>
<td>2.2468</td>
<td>2.3818</td>
<td>13.5908</td>
</tr>
</tbody>
</table>

*Not available.

namely PHS and IPHS), IP, MP, and BDA. Note that some of the problems have not been solved by all algorithms. For instance, the pressure vessel design problem has not been solved with BDA or MP as far as we know. Similarly the spring compression problem has not been solved with IP and HSA. These gaps in the computational results found in the literature on the considered problems are not substantial to hinder our experimental work and conclusions. Indeed, the recorded evidence in the large majority of cases points to the overwhelming superiority of PPA. Having said that, it must be added that further improvements to PPA and testing are being carried out on a more extensive collection of test problems including discrete ones.

Appendices

A. Welded Beam Design Optimisation

The welded beam design is a standard test problem for constrained design optimisation [6, 22]. There are four design variables: the width \(w\) and length \(L\) of the welded area, the depth \(d\) and thickness \(h\) of the main beam. The objective is to minimise the overall fabrication cost, under the appropriate constraints of shear stress \(\tau\), bending stress \(\sigma\), buckling load \(P\), and maximum end deflection \(\delta\). The optimization model is summarized as follows, where \(x^T = (w, L, d, h)\):

Minimise \(f(x) = 1.10471w^2L + 0.04811dh(14.0 + L)\),

subject to

\(g_1(x) = w - h \leq 0\),
\(g_2(x) = \delta(x) - 0.25 \leq 0\),
\(g_3(x) = \tau(x) - 13,600 \leq 0\),
\(g_4(x) = \sigma(x) - 30,000 \leq 0\),
\(g_5(x) = 1.10471w^2 + 0.04811dh(14.0 + L) - 5.0 \leq 0\),
\(g_6(x) = 0.125 - w \leq 0\),
\(g_7(x) = 6000 - P(x) \leq 0\),

where

\[\sigma(x) = \frac{504,000}{hd^2}, \quad D = \frac{1}{2}\sqrt{L^2 + (w + d)^2}, \quad Q = 6000\left(14 + \frac{L}{2}\right)\]
\[\delta = \frac{65,856}{30,000hd^2},\]
\[J = \sqrt{2wL}\left(\frac{L^2}{6} + \frac{(w + d)^2}{2}\right), \quad \alpha = \frac{6000}{\sqrt{2wL}},\]
\[\beta = \frac{QD}{J}, \quad P = 0.61423 \times 10^6 dh^3\frac{\delta}{6}(1 - \frac{\sqrt{30/48}}{28}), \quad \tau(x) = \sqrt{\frac{\alpha^2}{D} + \beta^2}.
\]

The simple limit or bounds are \(0.1 \leq L, d \leq 10\) and \(0.1 \leq w, h \leq 2.0\).

B. Pressure Vessel Design Optimisation

Pressure vessels are widely used in our daily life, such as champagne bottles and gas tanks [6, 39]. For a given volume and working pressure, the basic aim of designing a cylindrical vessel is to minimize the total cost. Typically, the design variables are the thickness \(d_1\) of the head, the thickness \(d_2\) of the body, the inner radius \(r\), and the length \(L\) of the cylindrical section [6]. This is a well-known test problem for optimization, where \(x^T = (d_1, d_2, r, L)\), and it can be written as

Minimise \(f(x) = 0.6224d_1rL + 1.7781d_2r^2 + 3.1661d_2^1L + 19.84d_1^2r\),

subject to

\(g_1(x) = -d_1 - 0.0193r \leq 0\),
\(g_2(x) = -d_2 - 0.00954r \leq 0\),
\(g_3(x) = -\pi r^2L - \frac{4\pi r^3}{3} + 129600 \leq 0\),
\(g_4(x) = -L - 240 \leq 0\).

The simple limits on the design variables are

\[0.0625 \leq d_1, d_2 \leq 99 \times 0.0625,\]
\[10.0 \leq r, L \leq 200.\]

C. Spring Design Optimisation

The main objective of this problem [33, 34] is to minimize the weight of a tension/compression spring, subject to constraints of minimum deflection, shear stress, surge frequency, and limits on outside diameter and on design variables. There are three design variables: the wire diameter \(x_1\), the mean coil diameter \(x_2\), and the number of active coils \(x_3\) [6].
Table 8: Optimal solutions for a variation on Himmelblau’s function.

<table>
<thead>
<tr>
<th>Solution vector</th>
<th>GA [38]</th>
<th>HSA [29]</th>
<th>PHS [27]</th>
<th>IPHS [27]</th>
<th>PPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>(x_1)</td>
<td>(\ast)</td>
<td>78.0</td>
<td>78.0</td>
<td>78.0</td>
<td>78.0</td>
</tr>
<tr>
<td>(x_2)</td>
<td>(\ast)</td>
<td>33.0</td>
<td>33.0</td>
<td>33.0</td>
<td>33.0</td>
</tr>
<tr>
<td>(x_3)</td>
<td>(\ast)</td>
<td>29.995</td>
<td>30.0053</td>
<td>29.9953</td>
<td>29.9952</td>
</tr>
<tr>
<td>(x_4)</td>
<td>(\ast)</td>
<td>45.0</td>
<td>45.0</td>
<td>45.0</td>
<td>45.0</td>
</tr>
<tr>
<td>(x_5)</td>
<td>(\ast)</td>
<td>36.776</td>
<td>36.7521</td>
<td>36.7756</td>
<td>36.7758</td>
</tr>
<tr>
<td>(g_1(x))</td>
<td>(\ast)</td>
<td>(\ast)</td>
<td>91.9963</td>
<td>91.9999</td>
<td>92.0</td>
</tr>
<tr>
<td>(g_2(x))</td>
<td>(\ast)</td>
<td>(\ast)</td>
<td>98.8362</td>
<td>98.8404</td>
<td>98.8405</td>
</tr>
<tr>
<td>(g_3(x))</td>
<td>(\ast)</td>
<td>(\ast)</td>
<td>20.0</td>
<td>20.0</td>
<td>20.0</td>
</tr>
<tr>
<td>(f(x))</td>
<td>-30665.500</td>
<td>-30665.500</td>
<td>-30663.845</td>
<td>-30665.533</td>
<td>-30665.540</td>
</tr>
</tbody>
</table>

\(\ast\)Not available.

The mathematical formulation of this problem, where \(x^T = (x_1, x_2, x_3)\), is as follows:

Minimize \(f(x) = (x_3^2 + 2)x_2x_1^2\),

subject to \(g_1(x) = 1 - \frac{x_3^2x_3}{7178x_1^4} \leq 0\),

\(g_2(x) = \frac{4x_2^2 - x_1x_2}{12,566(x_2x_1^2)} - x_1^2\) (C.1)

\(g_3(x) = 1 - \frac{140.45x_3}{x_2^2x_3} \leq 0\),

\(g_4(x) = \frac{x_3 + x_1}{1.5} - 1 \leq 0\).

The simple limits on the design variables are \(0.05 \leq x_1 \leq 2.0, 0.25 \leq x_2 \leq 1.3, \text{and } 2.0 \leq x_3 \leq 15.0\).

D. Speed Reducer Design Optimization

The problem of designing a speed reducer [40] is a standard test problem. It consists of the design variables as face width \(x_1\), module of teeth \(x_2\), number of teeth on pinion \(x_3\), length of the first shaft between bearings \(x_4\), length of the second shaft between bearings \(x_5\), diameter of the first shaft \(x_6\), and diameter of the first shaft \(x_7\) (all variables continuous except \(x_3\) that is integer). The weight of the speed reducer is to be minimized subject to constraints on bending stress of the gear teeth, surface stress, transverse deflections of the shafts, and stresses in the shaft [6]. The mathematical formulation of the problem, where \(x^T = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)\), is as follows:

Minimise \(f(x) = 0.7854x_1x_2^2 (3.3333x_3^2 + 14.9334x_3x_4)+ 14.9334x_3x_4 - 1.508x_1(x_4 + x_7) + 7.4777(x_5 + x_7) + 0.7854(x_4x_6^2 + x_5x_7^2),\)

subject to \(g_1(x) = \frac{27}{x_1x_2^2x_3} - 1 \leq 0\),

\(g_2(x) = \frac{397.5}{x_1x_2^2x_3} - 1 \leq 0\),

\(g_3(x) = \frac{1.93x_3}{x_2x_3x_4} - 1 \leq 0\),

\(g_4(x) = \frac{1.93x_3^2}{x_2x_3x_7} - 1 \leq 0\),

\(g_5(x) = \frac{1.0}{110x_5^3} \left(\frac{745.0x_4}{x_2x_3}\right)^2 + 16.9 \times 10^6 - 1 \leq 0\),

\(g_6(x) = \frac{1.0}{85x_7^3} \left(\frac{745.0x_5}{x_2x_3}\right)^2 + 157.5 \times 10^6 - 1 \leq 0\),

\(g_7(x) = \frac{x_2x_3}{40} - 1 \leq 0\),

\(g_8(x) = \frac{5x_3}{x_1} - 1 \leq 0\),

\(g_9(x) = \frac{x_1}{12x_2} - 1 \leq 0\),

\(g_{10}(x) = \frac{1.5x_6 + 1.9}{x_4} - 1 \leq 0\),

\(g_{11}(x) = \frac{1.1x_7 + 1.9}{x_5} - 1 \leq 0\) (D.1)

The simple limits on the design variables are

\(2.6 \leq x_1 \leq 3.6, \quad 0.7 \leq x_2 \leq 0.8, \quad 17 \leq x_3 \leq 28, \quad 7.3 \leq x_4 \leq 8.3, \quad 7.8 \leq x_5 \leq 8.3, \quad 2.9 \leq x_6 \leq 3.9, \quad 5.0 \leq x_7 \leq 5.5\) (D.2)
E. Constrained Optimization Problem 1

Minimize \[ f(x) = (x_1 - 2)^2 + (x_2 - 1)^2, \]
subject to \[ g_1(x) = x_1 - 2x_2 + 1 = 0, \]
\[ g_2(x) = \frac{-x_1}{4} - x_2^2 + 1 \geq 0, \]
\[ -10 \leq x_1, x_2 \leq 10. \]

F. Constrained Optimization Problem 2

Minimize \[ f(x) = (x_1 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2, \]
subject to \[ g_1(x) = 4.84 - (x_1 - 0.05)^2 \]
\[ - (x_2 - 2.5)^2 \geq 0, \]
\[ g_2(x) = x_1^2 + (x_2 - 2.5)^2 - 4.84 \geq 0, \]
\[ 0 \leq x_1, x_2 \leq 6. \]

G. Himmelblau’s Optimization Problem

Minimize \[ f(x) = 5.357847x_1^2 + 0.8356891x_1x_5 \]
\[ + 37.293239x_1 - 40792.141, \]
subject to \[ g_1(x) = 85.334407 + 0.0056858x_1x_5 \]
\[ + 0.00026x_1x_4 - 0.0022053x_3x_5, \]
\[ g_2(x) = 80.51249 + 0.0071341x_2x_5 \]
\[ + 0.0029955x_1x_3 + 0.0021813x_3^2, \]
\[ g_3(x) = 9.300961 + 0.0047026x_3x_5 \]
\[ + 0.0012547x_1x_3 + 0.0019085x_3x_4, \]
\[ 0 \leq g_1(x) \leq 92, \quad 90 \leq g_2(x) \leq 110, \]
\[ 20 \leq g_3(x) \leq 25, \quad 78 \leq x_1 \leq 102, \]
\[ 33 \leq x_2 \leq 45, \quad 27 \leq x_i \leq 45, \quad i = 3, 4, 5. \]

Conflict of Interests

The authors have no conflict of interests.

Acknowledgments

This work is supported by Abdul Wali Khan University, Mardan, Pakistan, Grant no. F:16-5/P& D/AWKUM/238.


Submit your manuscripts at http://www.hindawi.com