The Plant Propagation Algorithm for Discrete Optimisation: The Case of the Travelling Salesman Problem

Birsen İ. Selamoğlu and Abdellah Salhi

Abstract The Plant Propagation algorithm (PPA), has been demonstrated to work well on continuous optimization problems. In this paper, we investigate its use in discrete optimization and particularly on the well known Travelling Salesman Problem (TSP). This investigation concerns the implementation of the idea of short and long runners when searching for Hamiltonian cycles in complete graphs. The approach uses the notion of k-optimality. The performance of the algorithm on a standard list of test problems is compared to that of the Genetic Algorithm (GA), Simulated Annealing (SA), Particle Swarm Optimization (PSO) and the New Discrete Firefly Algorithm (New DFA). Computational results are included.

Keywords Discrete optimization · Plant Propagation Algorithm · Heuristics · Nature-inspired algorithms · Travelling Salesman Problem

1 Introduction

Real world problems are often hard to solve. Nature has evolved ways to deal with problems over millions of years. Today, when standard mathematical methods do not work, Scientists turn to Nature for inspiration. Hence the rapid development in the so called Nature-inspired algorithms or heuristics, [8]. Heuristics, generally, do not guarantee to find the optimum solution. However, they are often able to find good approximate solutions, in reasonable time, [55]. Some of the well-known algorithms in this class are the Genetic Algorithm [15], Simulated Annealing [23], Tabu Search [10], Ant Colony Optimization [6] and Particle Swarm Optimization [22]. Despite a long list of such algorithms, the complexity of problems that arise in daily applications, their size and the need to solve them in near real-time means that more robust and more efficient algorithms are required, [41]. Examples of algorithms which have been introduced recently are the Artificial Bee Colony algorithm, [19],

B.İ. Selamoğlu (✉) · A. Salhi
University of Essex, Wivenhoe Park, Colchester CO4 3SQ, UK
e-mail: bisela@essex.ac.uk

A. Salhi
e-mail: as@essex.ac.uk

X.-S. Yang (ed.), *Nature-Inspired Computation in Engineering*,
Studies in Computational Intelligence 637, DOI 10.1007/978-3-319-30235-5_3

the Firefly Algorithm [54], Cuckoo Search [57], the Bat Algorithm [56] and the Game Theory-based Multi-Agent approach, [44].

This paper is concerned with the Plant Propagation Algorithm or PPA, which implements the way the strawberry plant propagates; PPA is also referred to as the Strawberry Algorithm, [41]. It has been shown to work well on continuous unconstrained and constrained optimization problems, [47–49]. Here, we intend to establish that it can also work on discrete optimization problems. The specific problem under consideration is the Travelling Salesman Problem (TSP).

The rest of the paper is organised as follows. In Sect. 2, the TSP is defined and briefly reviewed. PPA is explained in Sect. 3 and its extension to handle discrete optimization problems is given in Sect. 4. Computational results and conclusion are given in Sect. 5.

2 TSP: A Brief Review

Combinatorial optimization is frequently met in real applications. Combinatorial or discrete optimisation problems are often computationally demanding. They more often than not belong to the so called NP-hard class of problems, [3, 9]. As such, it is not reasonable to expect exact solutions when solving large and practical instances. Thus, approximation methods, heuristics and meta-heuristics, are almost the norm when it comes to solving them, [35, 53].

A notoriously difficult and yet easy to state representative of this class of problems is the well known Travelling Salesman Problem, or TSP, [25]. The aim is to find the Hamiltonian cycle of shortest length in the complete weighted graph that represents a fully connected set of cities where the edges represent the connections between each pair of cities; an edge weight is the distance (time, cost …) that separates a pair of cities linked by the edge. Depending on the properties of these weights, we get different types of TSP. When the weights $c_{i,j}$ are put together in a square matrix $C = c_{i,j}, \forall\, i, j$ and $c_{i,j} = c_{j,i}, \forall\, i, j$, then we have a symmetric TSP or STSP, by virtue of matrix C which is symmetric. It is asymmetric if this property does not hold. If entries of C fulfil the triangle inequality, i.e. $c_{i,k} \leq c_{i,j} + c_{j,k}, \forall\, i, j, k$, the TSP is called metric. When $c_{i,j}$ is given as the Euclidean distances between nodes, the TSP is said to be Euclidean, [4, 25].

There are various methods to solve the TSP. As for other intractable combinatorial optimisation problems, exact algorithms are available, but only for relatively small instances of TSP; they do not work for large instances for efficiency reasons. Therefore, many heuristic approaches have been proposed in this respect, [14, 40].

Some of the exact algorithms for TSP are the Held-Karp algorithm which is based on dynamic programming [12], branch and bound [24], and branch and cut algorithms [33]. Some of the popular heuristic algorithms are Lin-Kernighan Local Search [27] and the strip algorithm [51]. Metaheuristics have become popular since they are able to find near optimal results in reasonable time even for large instances [38, 43, 58]. These are Nature-inspired algorithms which are now widely used to solve both continuous and discrete optimization problems. Most of these have been

applied to TSP. Early examples are the Genetic Algorithm (GA) [11], Simulated Annealing (SA) [1], Ant Colony Optimisation (ACO) [7], Discrete Particle Swarm Optimisation (DPSO) [5, 45] and Tabu Search (TS) [30].

In the following we review some of the most recently introduced Nature-inspired algorithms. Tsai et al. [52] have developed an algorithm called Heterogeneous Selection Evolutionary Algorithm (HeSEA). The algorithm has been tested on 16 TSP benchmark problems ranging from 318 to 13509-city problems. The algorithm is able to find the optimum result for up to 3038-city problem. The average errors are 0.05, 0.01 and 0.74 % for problems that have 4461, 5915 and 13509 cities, respectively. Song and Yang [46] have proposed an improved ACO using dynamic pheromone updating and a mutation strategy in order to increase the quality of the original algorithm. The new approach has given better results than the classical ACO and has found even better results than the best known solutions for some TSPs. Marinakis et al. [31] have proposed a hybrid algorithm to solve the Euclidean TSP problem. Their approach combines Honey Bees Mating Optimization algorithm (HBMOTSP), the Multiple Phase Neighborhood Search-Greedy Randomized Adaptive Search Procedure (MPNS-GRASP) and the Expanding Neighborhood Search Strategy. Experiments have been run on 74 benchmark TSP instances and have given competitive results.

Karaboga and Gorkemli [20] implemented the Combinatorial Artificial Bee Colony algorithm (CABC). They have adapted the Greedy Sub Tour Mutation (GSTM) operator proposed by Albayrak and Allahverdi [2], which increases the capability of GA to find the shortest length in TSP. The algorithm was used to solve two TSP instances with 150 and 200 cities, respectively. In [21], Gorkemli et al. have introduced the Quick Combinatorial Artificial Bee Colony Algorithm (qCABC) and improved CABC by changing the behaviour of onlooker bees. The new algorithm was tested against 9 heuristic methods including the CABC on same instances. The qCABC outperforms all algorithms except CABC on the 150-city problem. In [26], Li et al. have developed a Discrete Artificial Bee Algorithm (DABC) and applied it to TSP. They used a Swap Operator to represent the basic ABC for discrete problems. The performance of the algorithm was compared to that of PSO algorithm. Experimental results show that DABC outperforms PSO.

Jati and Suyanto [17] have introduced the Evolutionary Discrete Firefly Algorithm (EDFA) and tested it against the Memetic Algorithm (MA). EDFA was found to be better than MA on TSP instances. An improved version of EDFA was developed later by Jati et al. [16]; it uses a new movement scheme. This new version has outperformed the previous one in terms of efficiency.

Ouaarab et al. have introduced the Improved Cuckoo Search (ICS) and also two types of Discrete Cuckoo Search (DCS) algorithms, one of which is based on the original CS and the other one on the improved CS, to solve the TSP. The performance of the improved DCS was tested on 41 problems with the number of cities ranging from

Table 1 A compilation of recent notable results

Authors	Year	Algorithm	TSP size	Avg. error (%)
Song et al.	2006	Improved ACO	48–250 cities	0 % for all problems
Marinakis et al.	2011	HBMOTSP	51–85,900 cities	0 % for 63 out of 74 instances, less than 1 % for 11 instances
Karaboga et al.	2011	CABC	150 and 200 cities	0.9 % and 0.6 % respectively
Li et al.	2011	DABC	14–130 cities	Changing from 0.55 to 6.41 %
Jati et al.	2011	EDFA	16–666 cities	0 % up to 225-city instances and the 666-city problem, less than 12 % for instances of 225, 280 and 442 cities
Karaboga et al.	2013	qCABC	150 and 200 cities	0.7 % and 0.5 % respectively
Jati et al.	2013	New EDFA	16–666 cities	0 % up to 225-city instances and the 666-city problem, less than 12 % for instances of 225, 280 and 442 cities
Ouaarab et al.	2014	DCS	51–1379 cities	0 % for 13 out of 41 instances, less than 4.78 % as the worst for the 1379-city instance
Saenphon et al.	2014	FOGS-ACO	48–200 cities	0 % for the instance of 51-city, changing from 0.062 to 1.64 % for the other instances
Mahi et al.	2015	PSO-ACO-3Opt	51–200 cities	Changing from 0.00 to 0.95 %

51 to 1379. The experimental results show that the proposed method is superior to Genetic Simulated Annealing Ant Colony System with Particle Swarm Optimization Technique (GSA-ACS-PSOT) and DPSO [32]. Saenphon et al. [39] have developed the Fast Opposite Gradient Search method and combined it with ACO. The proposed method has been compared to TS, GA, PSO, ACO, PS-ACO and GA-PS-ACO on TSP instances. Mahi et al. have developed an algorithm based on PSO, ACO and 3-opt algorithms for TSP, [28]. The new hybrid method was tested against some well-known algorithms. The experimental results show that it outperforms the other

algorithms in terms of solution quality. The results of the new algorithm were considered to be satisfactory. Table 1 is a compilation of a set of results reported in fairly recent papers. The last column records the performance of the concerned algorithm on a set of TSP problems.

3 The Basic Plant Propagation Algorithm

The Plant Propagation Algorithm (PPA) introduced by Salhi and Fraga, [41], emulates the strategy that plants deploy to survive by colonising new places which have good conditions for growth. Plants, like animals, survive by overcoming adverse conditions using strategies. The strawberry plant, for instance, has a survival and expansion strategy which is to send short runners to exploit the local area if the latter has good conditions, and to send long runners to explore new and more remote areas, i.e. to run away from a not so favourable current area.

Algorithm 1 Pseudo-code of PPA, [41]

1: Generate a population $P = X_i,\ i = 1, \ldots,\ NP$;
2: $g \leftarrow 1$
3: **for** $g = 1 : g_{\max}$ **do**
4: Compute $N_i = f(X_i), \forall\ X_i \in P$
5: Sort P in ascending order of fitness values N (for minimization);
6: Create new population Φ
7: **for** each $X_i,\ i = 1, \ldots,\ NP$ **do**
8: $r_i \leftarrow$ set of runners where both the size of the set and the distance for each runner (individually) are proportional to the fitness values N_i;
9: $\Phi \leftarrow \Phi \cup r_i$ (append to population; death occurs by omission);
10: **end for**
11: $P \leftarrow \Phi$ (new population);
12: **end for**
13: **return** P, (the population of solutions).

The algorithm starts with a population of plants each of which represents a solution in the search space. X_i denotes the solution represented by plant i in an n-dimensional space. $X_i \in R^n$, i.e. $X_i = [x_{i,j}]$, for $j = 1, \ldots, n$ and $x_{ij} \in R$. NP is the population size, i.e. $i = 1, \ldots, n$ where $n_{\max}$ denotes the maximum number of runners that each plant can send. This iterative process stops when g the counter of generations reaches its given maximum value $g_{\max}$.

Individuals/plants/solutions are evaluated and then ranked (sorted in ascending or descending order) according to their objective (fitness) values and whether the problem is a min or a max problem. The number of runners of a plant is proportional to its objective value and conversely, the length of each runner is inversely proportional to the objective value, [41]. For each X_i, $N_i \in (0, 1)$ denotes the normalized objective function value. The number of runners for each plant to generate is

$$n_r^i = \lceil (n_{\max}\ N_i\ \beta_i) \rceil \tag{1}$$

where n_r^i shows the number of runners and $\beta_i \in (0, 1)$ is a randomly picked number. For each plant, the minimum number of runners is set to 1. The distance value found for each runner is denoted by dx_j^i. It is:

$$dx_j^i = 2(1 - N_i)(r - 0.5),\ for\ j = 1, \ldots, n. \tag{2}$$

where $r \in [0, 1]$ is a randomly chosen value.

Calculated distance values are used to position the new plants as follows:

$$y_{i,j} = x_{i,j} + (b_j - a_j)\ dx_j^i,\ for\ j = 1, \ldots, n. \tag{3}$$

where $y_{i,j}$ shows the position of the new plant and $[a_j, b_j]$ are the bounds of the search space.

The new population that is created by appending the new solutions to the current population is sorted. In order to keep the number of population constant, the solutions that have lower objective value are dropped.

4 Extension to Discrete Optimization Problems

PPA has been shown to work well on continuous unconstrained and constrained optimization problems, [47–49]. In this study, we consider the case of the Travelling Salesman Problem (TSP). The issues with the implementation of PPA to solve discrete optimization problems are:

1. Finding/Defining the equivalent of a distance between two solutions in the solution space which is a set of permutations representing tours.
2. Defining the neighbourhood of a solution, here a tour or permutation.

4.1 Implementation of PPA to Handle TSP

In any algorithm and in particular in population-based ones, representation of individuals/solutions is a key aspect of their implementation. The issue here is the representation of a plant which itself represents a solution. A solution here is any Hamiltonian cycle (tour) of the complete graph representation of the TSP. Note that representation affects the way the search/optimisation process as well as any stopping criteria are implemented.

4.1.1 The Representation of a Tour

A plant in the population of plants maintained by PPA is a tour/solution represented as a permutation of cities. X_i is tour i, $i = 1, \ldots, NP$. This means that the size of the population of plants is NP. Tours/plants are ranked according to their lengths. The tour length of plant i is denoted by N_i; it is a function of X_i. Without loss of generality, the Euclidean TSP is considered here. Tour lengths, therefore, are calculated according to the Euclidean distance

$$d_{x,y} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}. \tag{4}$$

In Fig.1, the entries of the array represent cities. City 1 and city 4 are successive in the depicted tour and the notation 1–4 defines the edge between them.

4.1.2 Distance Between Two Plants

One of the issues in implementing PPA is defining the distance that separates tours. Here it is defined as the number of exchanges to transform one tour into another. After sorting the tours by their tour lengths, a pre-determined number of the tours is taken amongst the ones that have good short lengths; short runners are then sent from these plants, i.e. new neighbouring tours are generated from them. The 2-opt rule is used for this purpose since it require the minimum number of changes to create new tours. The 2-opt move is implemented by removing two edges from the current tour and exchanging them with two other edges, [18].

An illustration of a 2-opt exchange can be seen in Fig.2. There, tour **a-b-d-c-a** has been transformed into **a-d-b-c-a** by exchanging edges **a-b** and **d-c**.

1	4	6	9	12	20	5	7	8	11	15	3	17	16	2	10	14	18	13	19

Fig. 1 The permutation representation of a plant as a tour

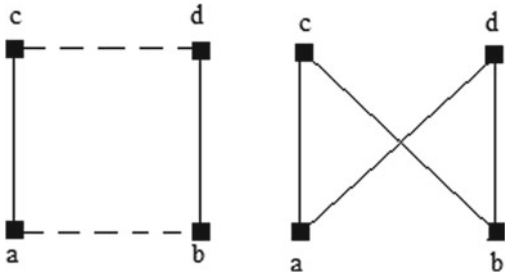

Fig. 2 2-opt exchange of a 4-city tour

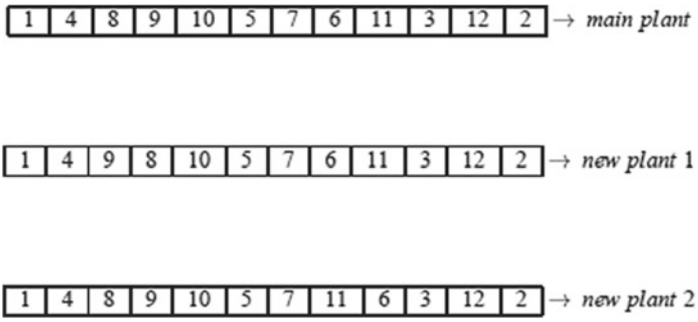

Fig. 3 Illustration of new plants generated by sending short runners from the main plant

Similarly, long runners are implemented by applying a k-opt rule with $k > 2$. In fact, this is pretty much the Lin-Kernighan algorithm (LK). It changes k edges in a tour, with k other edges. If, in this process, shorter tours are preferred and kept, then it will converge to potentially better solutions than it started with, [13].

This is not the only way available to measure the distance separating any two tours or permutations. However, it is in our view the best given the its efficiency and ease of implementation. An alternative approach can be found in [29], where a metric space of permutations defined using the k-opt rule has been studied.

4.1.3 Short and Long Runners

As mentioned earlier, in the basic PPA, a plant sends many short runners when it represents a good solution (exploitation move), or a few long runners when representing a poor solution (exploration move). Short runners are implemenetd using the 2-opt rule, and long runners, the k-opt rule, with $k > 2$.

An illustration of 2-opt rule implementing short runners can be seen in Fig. 3. In the figure, only two 2-opt neighbours of the main plant are shown. The first new plant was generated by exchanging the edges **4-8** and **8-9** and for the second one the edges **7-6** and **6-11** were exchanged. Note that the exchanged edges do not have to be adjacent.

Those tours in the population deemed to be representing poor solutions send one long runner each to explore the search space for better solutions. This is reasonable since a plant in a poor spot can hardly afford to send many long runners. Here, the long distance separating tours is implemented by applying a k-opt rule with $k > 2$. For both short and long runner cases, if the new tours have better results from these exchanges they are adopted and kept as new tours. Otherwise they are ignored.

An illustration of a new plant produced by sending a long runner can be seen in Fig. 4. The new plant is a 6-opt neighbour of the main plant. A 6-opt move can either be achieved by exchanging 6 edges chosen with other 6 edges or by implementing a number of 2-opt moves sequentially.

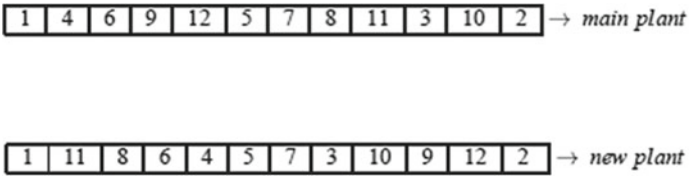

Fig. 4 Illustration of new plants generated by sending 1 long runner from the main plant

4.2 Pseudo-code of Discrete PPA

To the light of the general idea of implementing discrete PPA, we aimed at keeping the total computation time as short as possible. Therefore, it has been decided to start the algorithm with a good population of plants (tours). Diversity in the initial population is assumed to be guaranteed by the random processes used to generate tours. There are various such processes. Here, the initial population is generated using the greedy algorithm, random permutation or the strip algorithm, [38].

Algorithm 2 Pseudo-code of Discrete PPA

Generate a population $P = X_i,\ i = 1, \ldots, NP$ of valid tours; choose values for $g_{\max}$ and y.
g = 1
while g $< g_{\max}$ **do**
  Compute $N_i = f(X_i), \forall\ X_i \in P$
  Sort $N = N_i,\ i = 1, \ldots, NP$ in ascending order (for minimization);
  for $i = 1 : E(NP/10)$, Top 10 % of plants **do**
  Generate $\lceil (y/i) \rceil$ short runners for plant i using 2-opt rule, where y is an arbitrary parameter.
    if $N_i > f(r_i)$ **then**
      $X_i \leftarrow r_i$
    else
      Ignore r_i
    end if
  end for
  for $i = E(NP/10) + 1 : NP$ **do**
  $r_i = 1$ runner for plant i using k-opt rule, $k > 2$, 1 long runner for each plant not in the top 10 percent
    if $N_i > f(r_i)$ **then**
      $X_i \leftarrow r_i$
    else
      Ignore r_i
    end if
  end for
end while
return P, (the population of solutions).

Strip heuristics are preferred when solving large TSP instances, because they are simple to implement and relatively fast, [38]. The strip approach can be explained

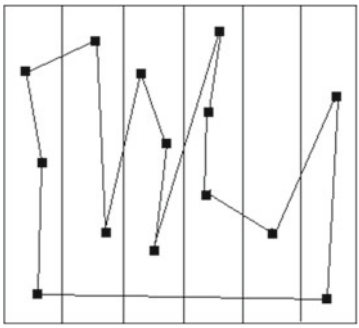

Fig. 5 An illustration of the Classical Strip Algorithm

as dividing the plane into vertical or horizontal strips, then grouping cities in each strip according to the order of their coordinates, ascending in some and descending in others, alternating between the two in the process. The length of the tour is calculated using the sum of the Euclidean distances between cities including that between the first and the last, [50]. An illustration of the strip algorithm can be seen in Fig. 5.

For short runners, the 2-opt algorithm has been implemented using speed-up techniques such as, "don't look bits", and "fixed radius search". Details of these mechanisms can be found in [18, 36]. For long runners, the 2-opt has been implemented sequentially many times in order to complete the number of exchanges a k-opt rule with $k > 2$ would achieve, [13].

5 Computational Results and Discussion

Discrete PPA has been implemented using the 2-opt rule to represent the concept of a short runner and a sequence of 2-opt moves to implement that of a long runner. Three sets of experiments have been carried out. The first set compares PPA with GA and SA. The second compares it with Modified PSO and the third with the New DFA. Each of these is discussed below.

5.1 PPA Versus GA and SA

The discrete implementation of PPA has been applied to 10 TSP instances ranging from 14 to 101 cities, [37]. These well known problem have also been solved with GA and SA elsewhere, in particular, in [42, 44]. The key parameters used in our experiments can be found in Table 2, below. For algorithms GA and SA, the relevant parameter values can be found in Tables 4 and 5.

Two termination criteria can be used: the first is the maximum number of generations set to 100, and the second is the number of iterations without any change in the

Table 2 Parameters used in PPA experimental results

Problem size	Population size	Max. gen[a]	Short runners	Long runners
14–51	40	100	10 (each a 2-opt)	1 (3 seq. 2-opts)
51–102	40	100	10 (each a 2-opt)	1 (4 seq. 2-opts)
102–666	100	100	10 (each a 2-opt)	1 (6 seq. 2-opts)

[a] Maximum number of generations
seq. = sequential

Table 3 Comparison between GA, SA and PPA for solving standard TSP instances, [37]

Problem	Optimum	GA		SA		Discrete PPA	
		Av. Dv. (%)	Av. Time (s)	Av. Dv. (%)	Av. Time (s)	Av. Dv. (%)	Av. Time (s)
burma14	30.8785	0.7	6.95	0.53	8.34	0	1.55
ulysses16	73.9876	0.27	8.26	0.17	42.28	0	2.71
ulysses22	75.3097	1.56	9.83	1.16	97.12	0	4.02
att48	33524	4.97	41.23	31.48	10.52	0.6	5.71
eil51	426	4.46	44.45	18.17	423.26	1.54	5.12
berlin52	7542	8.67	42.99	36.37	11.44	2.1	7.87
st70	675	11.62	66.17	24.89	232.21	1.66	9.23
eil76	538	6.84	74.9	33.34	1162.02	4.1	9.42
pr76	108,159	6.25	94.02	35.91	254.2	1.2	10.26
eil101	629	10.37	143.99	50.11	220.71	4.29	14.37

Table 4 Parameters of genetic algorithm

Population size	50
Maximum number of generations	20
The rate of crossover	0.95
The rate of mutation	0.075
The length of the chromosome	50 × 8-bits
The number of points of crossover	2

best tour found so far, which is set to 10. In these experiments, PPA outperformed both GA and SA on all instances. All algorithms have been coded in Matlab and each algorithm was run 5 times. Results of the comparison with GA and SA have been recorded in Table 3. Note that although the results are very encouraging, further testing on larger instances and comparison with other algorithms are needed to draw useful conclusions on performance. The really interesting aspects of PPA which have not been discussed yet are: (a) it is very simple to understand; (b) it involves less parameters than GA and SA, for instance.

Claim (a), above, is justified if we note that the algorithm is based on a universal principle which is "to stay in a favourable spot" (exploitation) and "to run away from an unfavourable spot" (exploration). That is all that the algorithm implements really.

Table 5 Parameters of simulated annealing

Maximum temperature	20
Minimum temperature	1
$\alpha\%$	0.5 %
P	5
Number of iterations at each temperature	20
Temperature set	[20,10,5,3,1]

But, that is all a global search algorithm requires too. Claim (b) is equally easy to justify. Let us consider the list of parameters that are arbitrarily set in GA.

1. The population size;
2. The maximum number of generations;
3. The number of generations without improvement to stop;
4. The rate of crossover;
5. The rate of mutation;
6. The length of the chromosome;
7. The number of points of crossover.

Now, compare the above list to that of PPA. We can start the algorithm with a single plant that will then produce more plants unlike GA where a population with more than one individual is required. The number of runners can be decided by the objective value of each plant; indeed in Nature, some plants may have no runners at all because they are in desperate conditions while other may have 1, 2 or more depending on where they are and the corresponding value they give to the objective function. If we accept this, then PPA requires no more than a mechanism to stop, i.e. a stopping criterion. Hence, the comparatively short list of its parameters as in Table 4.

1. The maximum number of generations;
2. The number of generations without improvement to stop;
3. The maximum number of runners any plant can have.

For ease of implementation more parameters are used.

Simulated Annealing is not as extravagant as GA when it comes to arbitrary parameters. However, it still requires at least five parameters.

1. The maximum temperature;
2. The minimum temperature;
3. Parameter α: Percentage improvement in the objective value expected in each move;
4. The maximum number of moves without achieving $\alpha\%$ of objective function value improvement (P);
5. Number of iterations at each temperature.

To this, one can add the temperature set as in the last row of Table 5.

It is, therefore, fair to say that PPA compares well against GA and SA even if only small instances of TSP have been considered. Note that the proliferation of arbitrary parameters makes the concerned algorithms less usable since it is difficult to find good default parameters when the list is long. More arbitrary parameters also mean more uncertainty. Based on this comparison approach, it is fair too to say that PPA will match most heuristics and hyper-heuristics. Further work may be to design a more realistic algorithm comparison methodology which not only takes into account raw performance, but also what is required in terms of parameter setting.

5.2 PPA Versus Modified PSO

A second set of experiments has been conducted and the results compared to those obtained by the Modified PSO, [34]. There are four versions of PSO was studied. All algorithms have been applied to 4 TSP instances with 14 to 76 cities, [37]. Each algorithm was run 10 times for each problem. The results of the comparison can be found in Table 6. The parameters values used in the PPA experiments can be found in Table 2. Those of PSO can be found in Table 7.

Table 6 Comparison between modified PSO and PPA for solving standard TSP instances, [37]

Problem	PSO-TS	PSO-TS-2opt	PSO-TS-CO	PSO-TS-CO-2opt	Discrete PPA
	Av. Dv. (%)	Av. Dv. (%)	Av. Dv. (%)	Av. Dv. (%)	Av. Dv. (%)
burma14	9.12	0	10	0	0
eil51	35.47	6.81	16.34	2.54	1.84
eil76	9.98	5.46	12.86	4.75	3.76
berlin52	7.37	5.22	10.33	2.12	1.84

PSO-TS : The PSO based on space transformation
PSO-TS-2opt: PSO-TS combined with 2-opt local search
PSO-TS-CO: PSO-TS with chaotic operations
PSO-TS-CO-2opt: PSO-TS combined with CO and 2-opt

Table 7 Parameters of modified PSO

The number of particles	50
The value of V_{max}	0.1
The values for learning factors, c1 and c2	c1 = c1 = 2
The inertia coefficient	1
P_{max}	1
Local search probability	0.01
Dispitive probability	0.001
The maximum number of generations	2000

In these experiments, PPA outperformed all Modified PSO algorithms on all instances in terms of solution quality. Arbitrarily set parameters for Modified PSO are as listed below, [34]:

1. The number of particles;
2. The value of $V_{\max}$;
3. The values for learning factors, c1 and c2;
4. The inertia coefficient;
5. A positive real number $P_{\max}$, to express the range of the activities for each particle;
6. Local search probability;
7. Dispitive Probability;
8. The maximum number of generations;
9. The number of generations without improvement to stop.

Figure 6a–d depict the tours found in generations 1, 3, 5 and 8 of PPA when applied to a 22-city instance [37]. The last figure shows the optimal tour. Figure 7a–d show the convergence curves of the algorithm on four TSP instances with the number of cities ranging from 48 to 225.

5.3 PPA Versus New DFA

Another set of experiments has been conducted and the results compared to those obtained by the Discrete Firefly Algorithm (New DFA) described in [16, 58]. Both algorithms have been applied to 7 TSP instances with 16 to 666 cities, [37]. The parameter values used in PPA experiments can be found in Table 2. For New DFA, they can be found in Table 9. Note that the average accuracy was calculated using the equation below.

$$Avg.\ accuracy = \frac{Best\ known\ solution}{Avg.\ solution\ found} * 100 \tag{5}$$

Each algorithm was run 50 times. The results of the comparison can be found in Tables 8 and 9.

In these experiments, in terms of solution quality PPA outperformed New DFA on 3 out of 7 instances. On the remaining 4 instances both algorithms have found the optimum solution. Arbitrarily set parameters required by New DFA are as listed below, [16]:

1. The number of maximum generations;
2. The population size;
3. The light absorption coefficient;
4. The updating index.

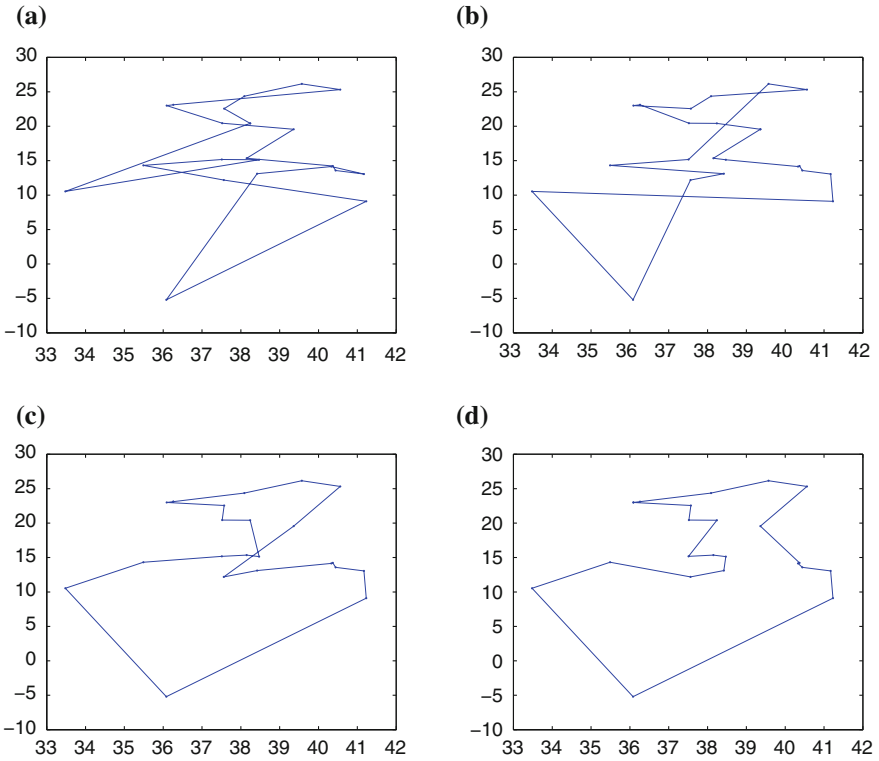

Fig. 6 **a** 22-city problem-1st Generation. **b** 22-city problem-3rd Generation. **c** 22-city problem-5th Generation. **d** 22-city problem-8th Generation

5.4 Conclusion and Further Investigation

We have shown that PPA can be implemented to handle discrete problems and in particular the TSP. The results, although limited given the sizes of the problems considered, are very encouraging and overall in favour of PPA. The idea of comparing algorithms/heuristics on other criteria than just performance is appealing in many respects; the criterion consisting of the number of arbitrarily set parameters required by a given algorithm/heuristic is interesting because it does not depend on the programming skills of the researcher, the quality of codes, the language of coding, the compiler, the processor etc. It is intrinsic to the algorithm/heuritic. Further research, therefore, could be to explore how the number of arbitrarily set parameters required by different algorithms/heuristics, affects their performance and distinguishes between them. It is also interesting to know if a parameter is at all necessary in the implementation of a heuristic. In the case of PPA, the number of plants is not really necessary to start with since the algorithm works perfectly well starting with a single plant.

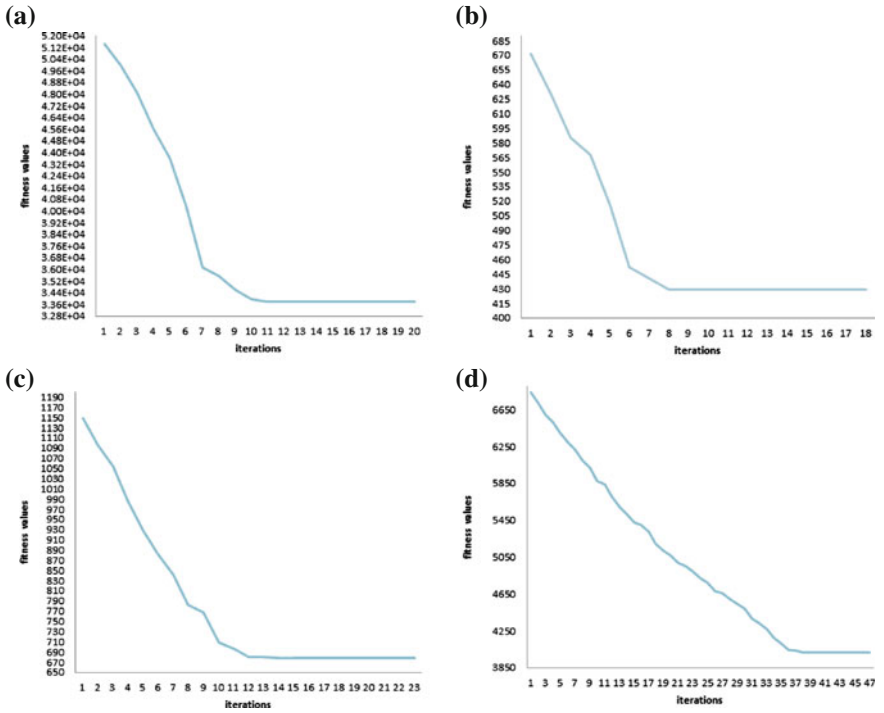

Fig. 7 **a** Curve evolution diagram of att48. **b** Curve evolution diagram of eil51. **c** Curve evolution diagram of st70. **d** Curve evolution diagram of tsp225

Table 8 PPA versus new DFA and PPA on standard TSP instances, [37]

Problem	Optimum	New DFA	PPA
		Av. Acc. (%)	Av. Acc. (%)
ulysses16	73.9876	100	100
ulysses22	75.3097	100	100
gr202	549.99	100	100
tsp225	3845	88.332	94.242
a280	2578	88.297	93.392
pcb442	50778	88.505	93.985
gr666	3952.53	100	100

Table 9 Parameters of new DFA

The number of maximum generations	Between 100–500 generations
The population size	5
The light absorption coefficient	0.001
The updating index	Between 1 and 16

Similar insights can be gained if this is considered when experimenting with a variety of heuristics.

Further investigation concerns extending PPA to solve other discrete problems such as, Knapsack, set covering and facility location problems. Results of this extension will be reported in future papers.

References

1. Aarts, E.H., Korst, J.H., van Laarhoven, P.J.: A quantitative analysis of the simulated annealing algorithm: a case study for the traveling salesman problem. J. Stat. Phys. **50**(1–2), 187–206 (1988)
2. Albayrak, M., Allahverdi, N.: Development a new mutation operator to solve the traveling salesman problem by aid of genetic algorithms. Expert Syst. Appl. **38**(3), 1313–1320 (2011)
3. Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M.: Proof verification and the hardness of approximation problems. J. ACM **45**(3), 501–555 (1998). doi:10.1145/278298.278306. http://doi.acm.org/10.1145/278298.278306
4. Bellmore, M., Nemhauser, G.L.: The traveling salesman problem: a survey. Oper. Res. **16**(3), 538–558 (1968)
5. Clerc, M.: Discrete particle swarm optimization, illustrated by the traveling salesman problem. In: New Optimization Techniques in Engineering, pp. 219–239. Springer (2004)
6. Dorigo, M.: Ant colony optimization and swarm intelligence. In: Proceedings of 5th International Workshop, ANTS 2006, Brussels, Belgium, September 4–7, 2006, vol. 4150. Springer Science & Business Media (2006)
7. Dorigo, M., Gambardella, L.M.: Ant colonies for the travelling salesman problem. BioSystems **43**(2), 73–81 (1997)
8. Fister Jr., I., Yang, X.S., Fister, I., Brest, J., Fister, D.: A brief review of nature-inspired algorithms for optimization. arXiv preprint arXiv:1307.4186 (2013)
9. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1979)
10. Glover, F., Laguna, M.: Tabu Search. Springer (1999)
11. Grefenstette, J., Gopal, R., Rosmaita, B., Van Gucht, D.: Genetic algorithms for the traveling salesman problem. In: Proceedings of the First International Conference on Genetic Algorithms and their Applications, pp. 160–168. Lawrence Erlbaum, New Jersey (1985)
12. Held, M., Karp, R.M.: A dynamic programming approach to sequencing problems. J. Soc. Ind. Appl. Math 196–210 (1962)
13. Helsgaun, K.: An effective implementation of k-opt moves for the lin-kernighan tsp heuristic. Ph.D. thesis, Roskilde University. Department of Computer Science (2006)
14. Hoffman, K.L., Padberg, M., Rinaldi, G.: Traveling salesman problem. In: Encyclopedia of Operations Research and Management Science, pp. 1573–1578. Springer (2013)
15. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. U Michigan Press (1975)
16. Jati, G.K., Manurung, R., Suyanto: 13–discrete firefly algorithm for traveling salesman problem: a new movement scheme. In: X.S.Y.C.X.H.G. Karamanoglu (ed.) Swarm Intelligence and Bio-Inspired Computation, pp. 295–312. Elsevier, Oxford (2013). doi:10.1016/B978-0-12-405163-8.00013-2. http://www.sciencedirect.com/science/article/pii/B9780124051638000132
17. Jati, G.K., Suyanto: Evolutionary Discrete Firefly Algorithm for Travelling Salesman Problem. In: Bouchachia, Abdelhamid (ed.) Adaptive and Intelligent SystemsSpringer, pp. 393–403.Springer, Berlin, Heidelberg (2011). ISBN:978-3-642-23857-4

18. Johnson, D.S., McGeoch, L.A.: The traveling salesman problem: a case study in local optimization. Local Search Comb. Optim. **1**, 215–310 (1997)
19. Karaboga, D., Basturk, B.: A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. J. Global Optim. **39**(3), 459–471 (2007)
20. Karaboga, D., Gorkemli, B.: A combinatorial artificial bee colony algorithm for traveling salesman problem. In: Innovations in Intelligent Systems and Applications (INISTA), 2011 International Symposium on, pp. 50–53. IEEE (2011)
21. Karaboga, D., Gorkemli, B.: A quick artificial bee colony-qabc-algorithm for optimization problems. In: Innovations in Intelligent Systems and Applications (INISTA), 2012 International Symposium on, pp. 1–5. IEEE (2012)
22. Kennedy, J.: Particle swarm optimization. In: Encyclopedia of Machine Learning, pp. 760–766. Springer (2010)
23. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983)
24. Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica: J. Econometric Soc. 497–520 (1960)
25. Laporte, G.: The traveling salesman problem: an overview of exact and approximate algorithms. Eur. J. Oper. Res. **59**(2), 231–247 (1992)
26. Li, L., Cheng, Y., Tan, L., Niu, B.: A discrete artificial bee colony algorithm for tsp problem. In: Bio-Inspired Computing and Applications, pp. 566–573. Springer (2012)
27. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. **21**(2), 498–516 (1973). doi:10.1287/opre.21.2.498. http://dx.doi.org/10.1287/opre.21.2.498
28. Mahi, M., mer Kaan Baykan, Kodaz, H.: A new hybrid method based on particle swarm optimization, ant colony optimization and 3-opt algorithms for traveling salesman problem. Appl. Soft Comput. 30, 484–490 (2015). doi:10.1016/j.asoc.2015.01.068. http://www.sciencedirect.com/science/article/pii/S1568494615000940
29. Mak, K.T., Morton, A.J.: Distances between traveling salesman tours. Discrete Appl. Math. **58**(3), 281–291 (1995). doi:10.1016/0166-218X(93)E0115-F. http://www.sciencedirect.com/science/article/pii/0166218X93E0115F
30. Malek, M., Guruswamy, M., Pandya, M., Owens, H.: Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Ann. Oper. Res. **21**(1), 59–84 (1989)
31. Marinakis, Y., Marinaki, M., Dounias, G.: Honey bees mating optimization algorithm for the euclidean traveling salesman problem. Inform. Sci. **181**(20), 4684–4698 (2011)
32. Ouaarab, A., Ahiod, B., Yang, X.S.: Improved and discrete cuckoo search for solving the travelling salesman problem. In: Cuckoo Search and Firefly Algorithm, pp. 63–84. Springer (2014)
33. Padberg, M., Rinaldi, G.: A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. SIAM Rev. **33**(1), 60–100 (1991)
34. Pang, W., Wang, K.P., Zhou, C.G., Dong, L.J., Liu, M., Zhang, H.Y., Wang, J.Y.: Modified particle swarm optimization based on space transformation for solving traveling salesman problem. In: Proceedings of 2004 International Conference on Machine Learning and Cybernetics, 2004, vol. 4, pp. 2342–2346. IEEE (2004)
35. Papadimitriou, C.H., Steiglitz, K.: Combinatorial Optimization: Algorithms and Complexity. Courier Corporation (1998)
36. Rego, C., Gamboa, D., Glover, F., Osterman, C.: Traveling salesman problem heuristics: leading methods, implementations and latest advances. Eur. J. Oper. Res. **211**(3), 427–441 (2011)
37. Reinelt, G.: TSPLIB–A T.S.P. library. Tech. Rep. 250, Universität Augsburg, Institut für Mathematik, Augsburg (1990)
38. Reinelt, G.: The Traveling Salesman: Computational Solutions for TSP Applications. Springer, Berlin (1994)
39. Saenphon, T., Phimoltares, S., Lursinsap, C.: Combining new fast opposite gradient search with ant colony optimization for solving travelling salesman problem. Eng. Appl. Artif. Intell. **35**,

324–334 (2014). doi:10.1016/j.engappai.2014.06.026. http://dx.doi.org/10.1016/j.engappai.2014.06.026
40. Salhi, A.: The ultimate solution approach to intractable problems. In: Proceedings of the 6th IMT-GT Conference on Mathematics, Statistics and its Applications, pp. 84–93 (2010)
41. Salhi, A., Fraga, E.: Nature-inspired optimisation approaches and the new plant propagation algorithm. In: Proceedings of the ICeMATH2011 pp. K2–1 to K2–8 (2011)
42. Salhi, A., Proll, L.: Rios Insua, D., Martin, J.: Experiences with stochastic algorithms for a class of constrained global optimisation problems. RAIRO–Oper. Res. 34, 183–197 (2000). doi:10.1051/ro:2000110
43. Salhi, A., Rodríguez, J.A.V.: Tailoring hyper-heuristics to specific instances of a scheduling problem using affinity and competence functions. Memetic Comput. **6**(2), 77–84 (2014)
44. Salhi, A., Töreyen, Ö.: A game theory-based multi-agent system for expensive optimisation problems. In: Computational Intelligence in Optimization, pp. 211–232. Springer (2010)
45. Shi, X.H., Liang, Y.C., Lee, H.P., Lu, C., Wang, Q.: Particle swarm optimization-based algorithms for tsp and generalized tsp. Inform. Process. Lett. **103**(5), 169–176 (2007)
46. Song, X., Li, B., Yang, H.: Improved ant colony algorithm and its applications in tsp. In: Sixth International Conference on Intelligent Systems Design and Applications, 2006. ISDA'06, vol. 2, pp. 1145–1148. IEEE (2006)
47. Sulaiman, M., Salhi, A.: A Seed-based plant propagation algorithm: the feeding station model. Sci World J (2015)
48. Sulaiman, M., Salhi, A., Fraga, E.S.: The Plant Propagation Algorithm: Modifications and Implementation. ArXiv e-prints (2014)
49. Sulaiman, M., Salhi, A., Selamoglu, B.I., Kirikchi, O.B.: A plant propagation algorithm for constrained engineering optimisation problems. Mathematical Problems in Engineering 627416, 10 pp (2014). doi:10.1155/2014/627416
50. Supowit, K.J., Plaisted, D.A., Reingold, E.M.: Heuristics for weighted perfect matching. In: Proceedings of the Twelfth Annual ACM Symposium on Theory of Computing, pp. 398–419. ACM (1980)
51. Supowit, K.J., Reingold, E.M., Plaisted, D.A.: The travelling salesman problem and minimum matching in the unit square. SIAM J. Comput. **12**(1), 144–156 (1983)
52. Tsai, H.K., Yang, J.M., Tsai, Y.F., Kao, C.Y.: An evolutionary algorithm for large traveling salesman problems. IEEE Trans. Syst. Man Cybern. Part B: Cybern. 34(4), 1718–1729 (2004)
53. Yagiura, M., Ibaraki, T.: On metaheuristic algorithms for combinatorial optimization problems. Syst. Comput. Jpn **32**(3), 33–55 (2001)
54. Yang, X.S.: Firefly algorithm, stochastic test functions and design optimisation. Int. J. Bio-Inspir. Comput. **2**(2), 78–84 (2010)
55. Yang, X.S.: Nature-inspired Metaheuristic Algorithms, 2nd edn. (2010)
56. Yang, X.S.: A new metaheuristic bat-inspired algorithm. In: Nature Inspired Cooperative Strategies for Optimization (NICSO 2010), pp. 65–74. Springer (2010)
57. Yang, X.S., Deb, S.: Cuckoo search via lévy flights. In: World Congress on Nature & Biologically Inspired Computing, 2009. NaBIC 2009. pp. 210–214. IEEE (2009)
58. Yang, X.S., Cui, Z., Xiao, R., Gandomi, A.H., Karamanoglu, M.: Swarm Intelligence and Bio-inspired Computation: Theory and Applications. Newnes (2013)