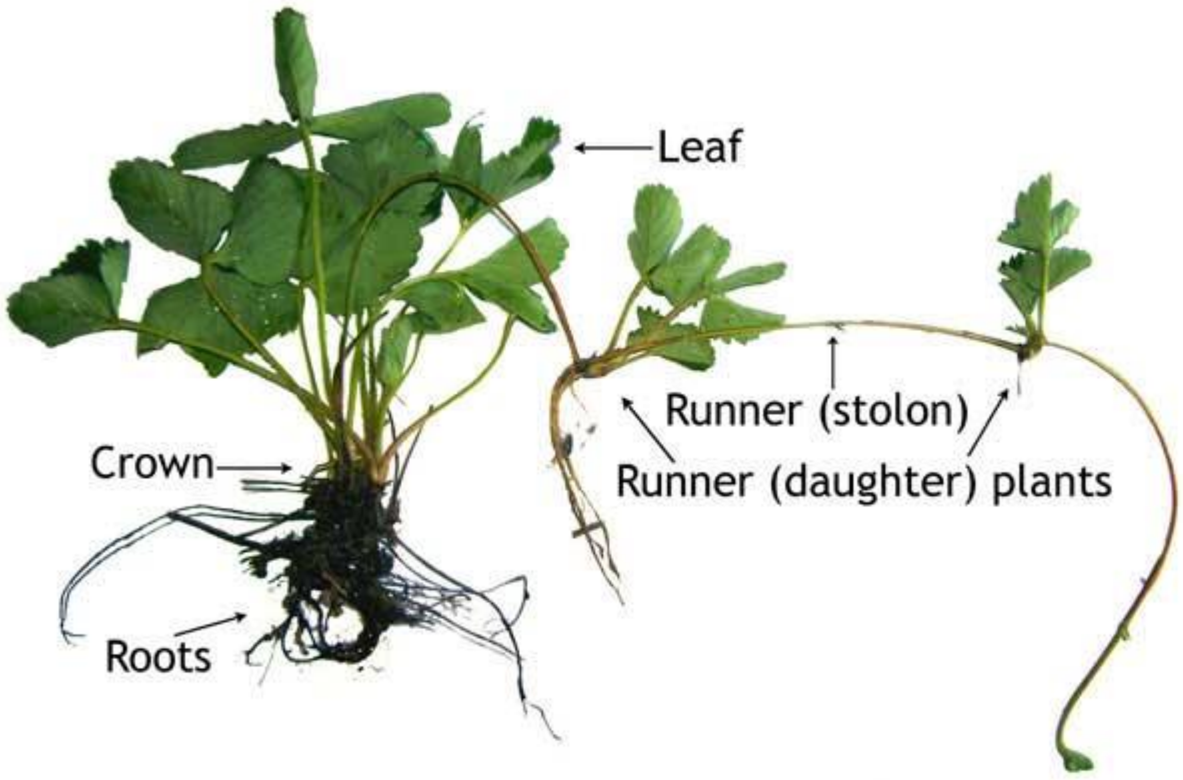
The Plant Propagation Algorithm for Discrete Optimization: The Case of the Travelling Salesman Problem

Birsen ˙I. Selamo˘glu and Abdellah Salhi
University of Essex,
Wivenhoe Park, Colchester CO4 3SQ, UK

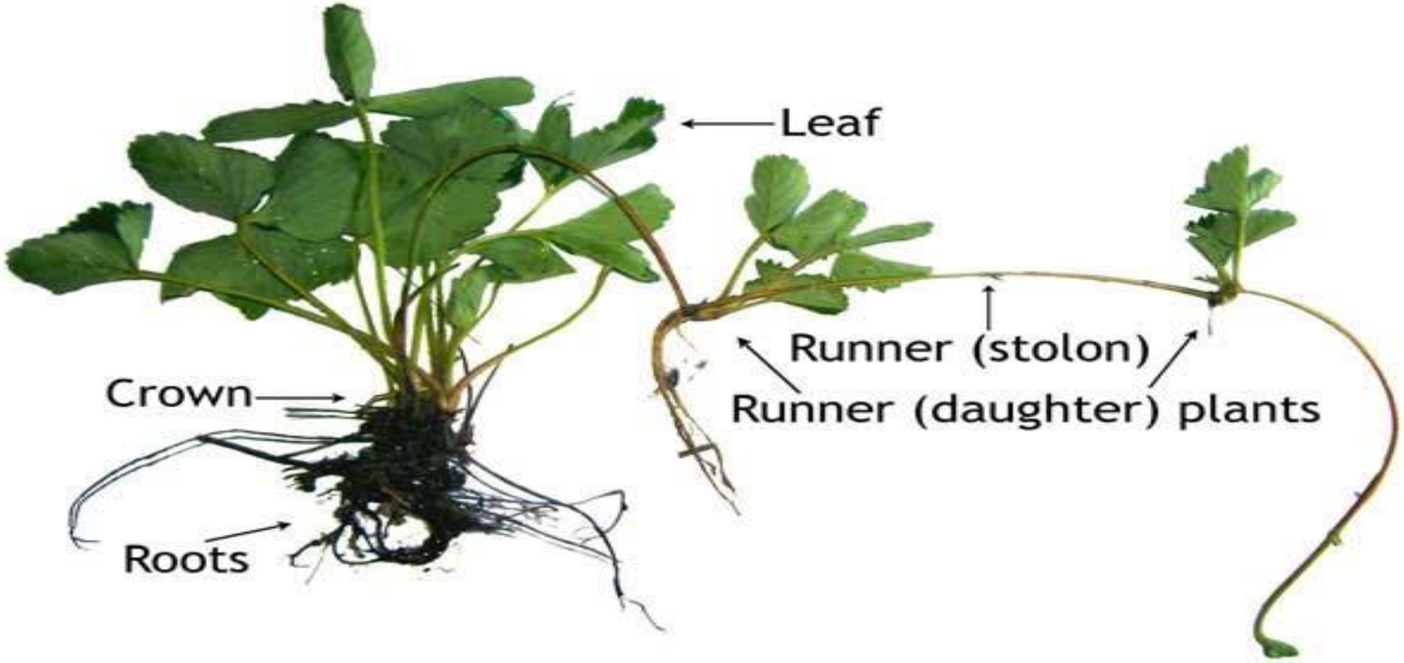


Presented by

Md. Siddiqur Rahman Tanveer
Roll: 1707505
Department of CSE, KUET

History..

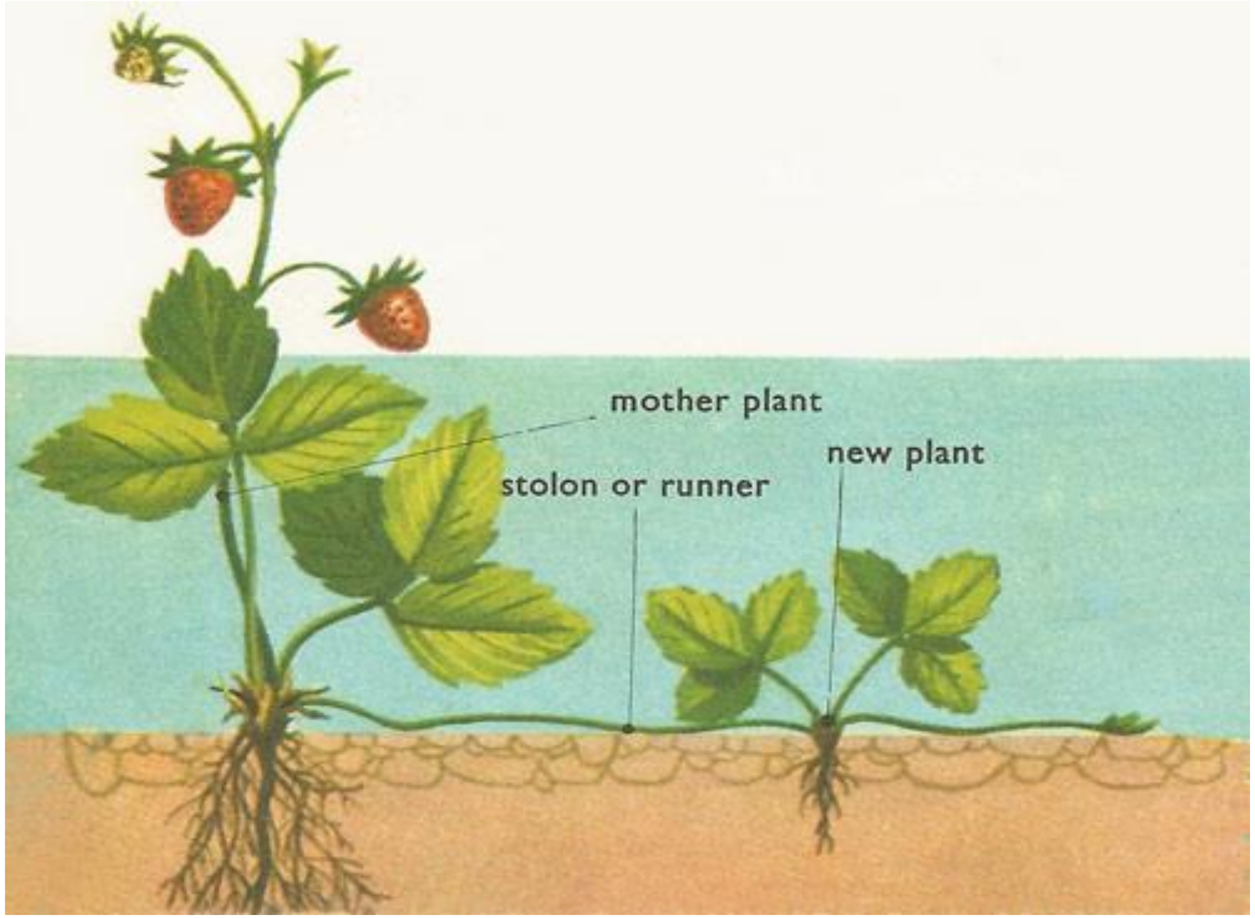
The Plant Propagation Algorithm (PPA) introduced by Salhi and Fraga, [1] emulates the strategy that plants deploy to survive by colonising new places which have good conditions for growth. Plants, like animals, survive by overcomeing adverse conditions using strategies.



Strawberry Plant Propagation

There are three main ways to propagate strawberry plants [2]

- The plants can be divided and transplanted once multiple crowns have been grown (or division of rhizomes)
- new plants can be grown from strawberry seeds
- or the runners that strawberry plants put out can be controlled, guided, and caused to root where clone plants can be utilized most efficiently.



Strawberry Plant Propagation

- The strawberry plant, for instance, has a survival and expansion strategy

Send **short runners**

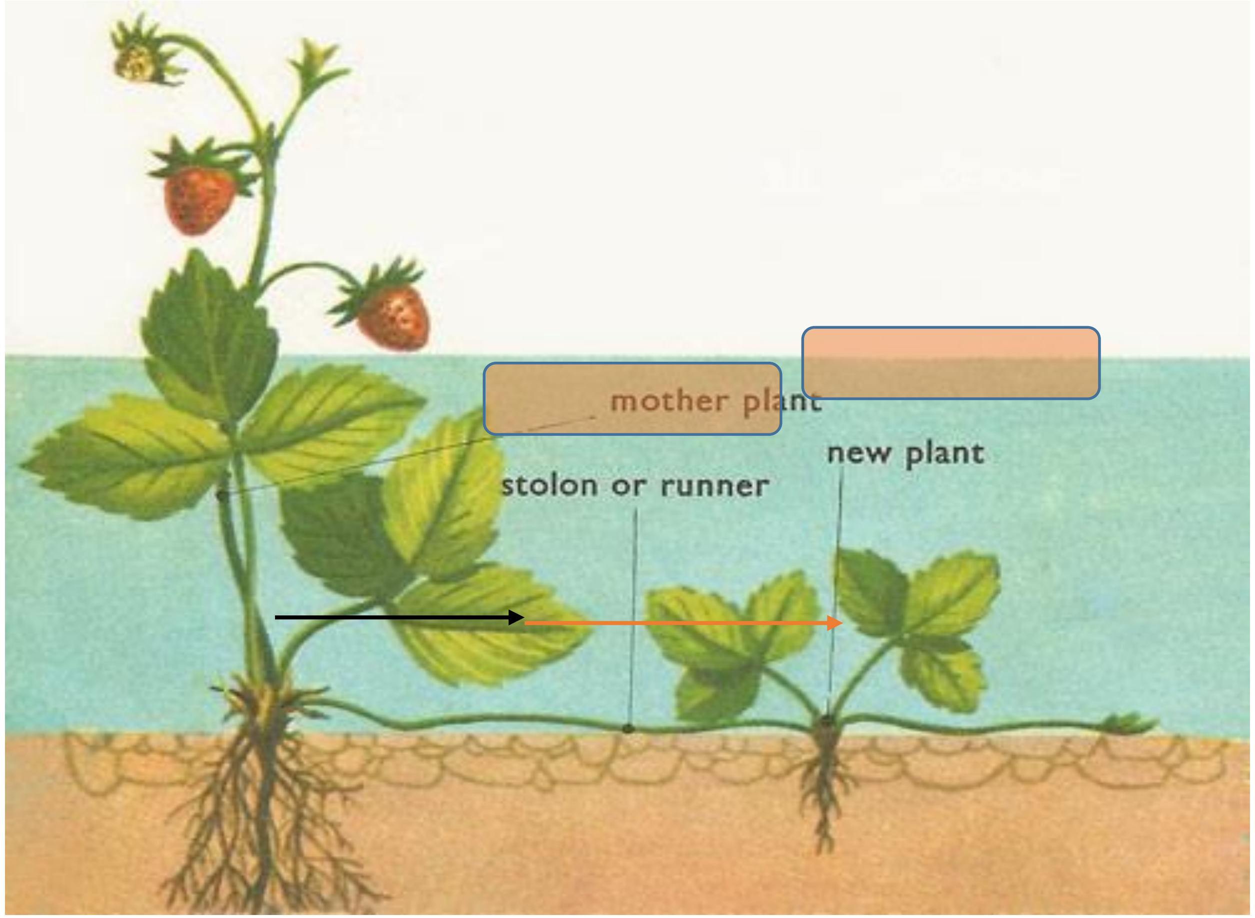
exploit local area

if the latter has good conditions

Send **send long runners**

explore new and more remote areas

i.e. to run away from a not so favorable current area



Plant Propagation Algorithm

The algorithm starts with a population of plants each of which represents a solution in the search space

Algorithm 1 Pseudo-code of PPA, [41]

Generate a population $P = X_i,\ i = 1, \ldots,\ NP$;
$g \leftarrow 1$
for $g = 1 : g_{\max}$ **do**
    Compute $N_i = f(X_i), \forall\ X_i \in P$
    Sort P in ascending order of fitness values N (for minimization);
    Create new population Φ
    for each $X_i,\ i = 1, \ldots,\ NP$ **do**
    $r_i \leftarrow$ 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;
    $\Phi \leftarrow \Phi \cup\ r_i$ (append to population; death occurs by omission);
    end for
    $P \leftarrow \Phi$ (new population);
end for
return P, (the population of solutions).

Key elements of Plant Propagation Algorithm

- $\boldsymbol{X_i}$ - 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 *Xi* , *Ni* ∈ *(*0, 1*)* denotes the normalized objective function value. The number of runners for each plant to generate is

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

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:

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

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

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

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

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

where *yi, j* shows the position of the new plant and [*aj* , *bj*] 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.

Travelling Salesman Problem

Given a set of cities and distance between every pair of cities, the problem is to find the shortest possible route that visits every city exactly once and returns to the starting point.

Extension to Discrete Optimization Problems

PPA has been shown to work well on continuous unconstrained and constrained optimization problems, [47–49]. In this study, they 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 neighborhood of a solution, here a tour or permutation.

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.

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.

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

Distance Between Two Plants

Without loss of generality, the Euclidean TSP is considered here. Tour lengths, therefore, are calculated according to the Euclidean distance

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

Short and Long Runners

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**.

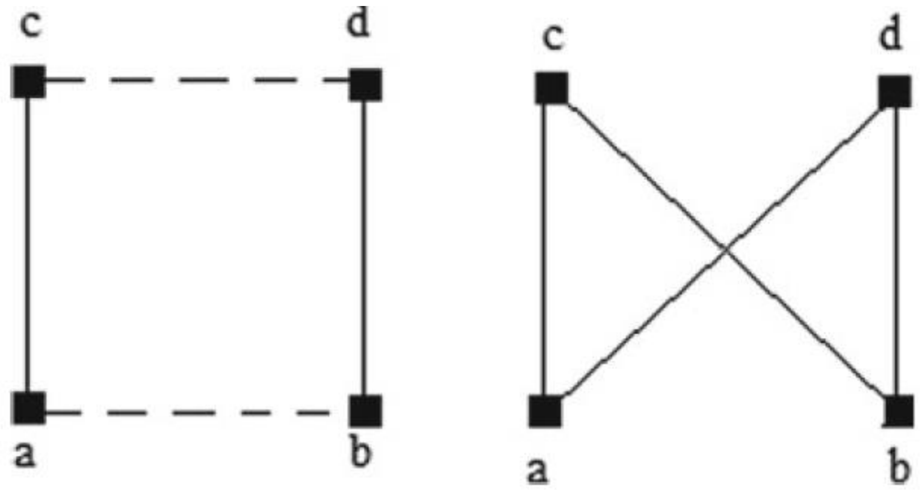


Fig. 2 2-opt exchange of a 4-city tour

New plants generated by sending short runners from the main plant

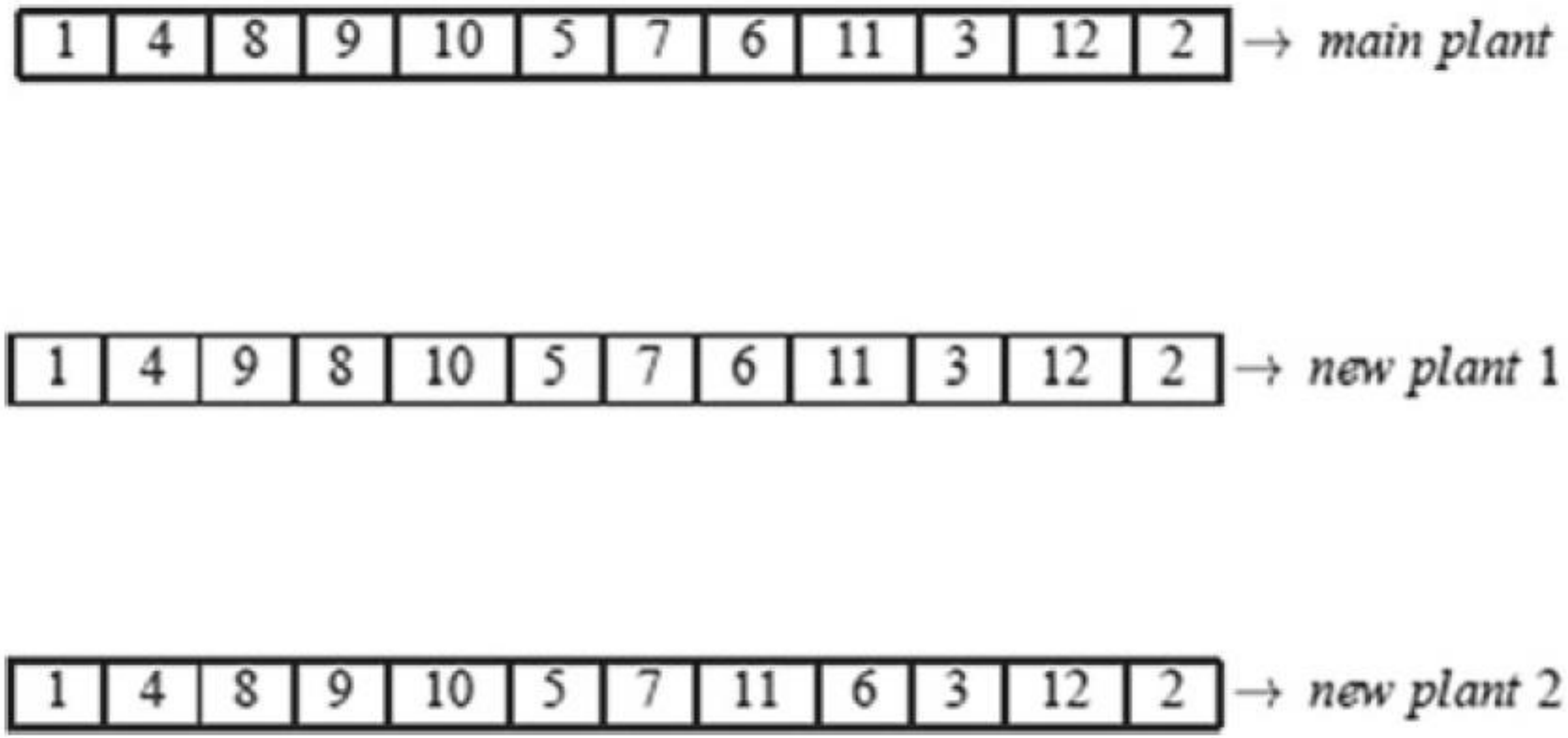


Fig. 3 Illustration of new plants generated by sending short runners from the main plant

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].

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.

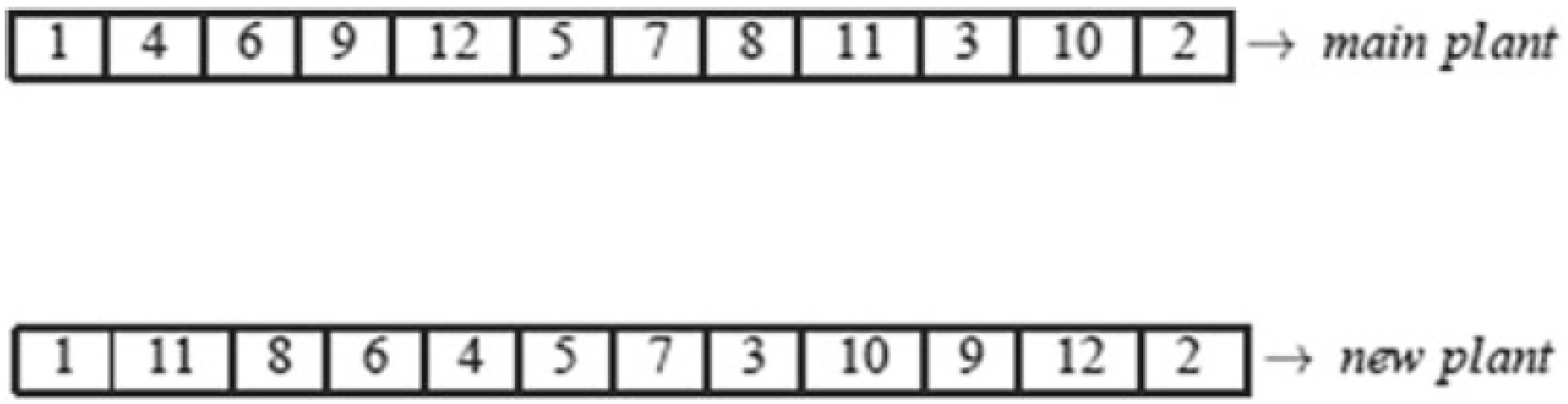
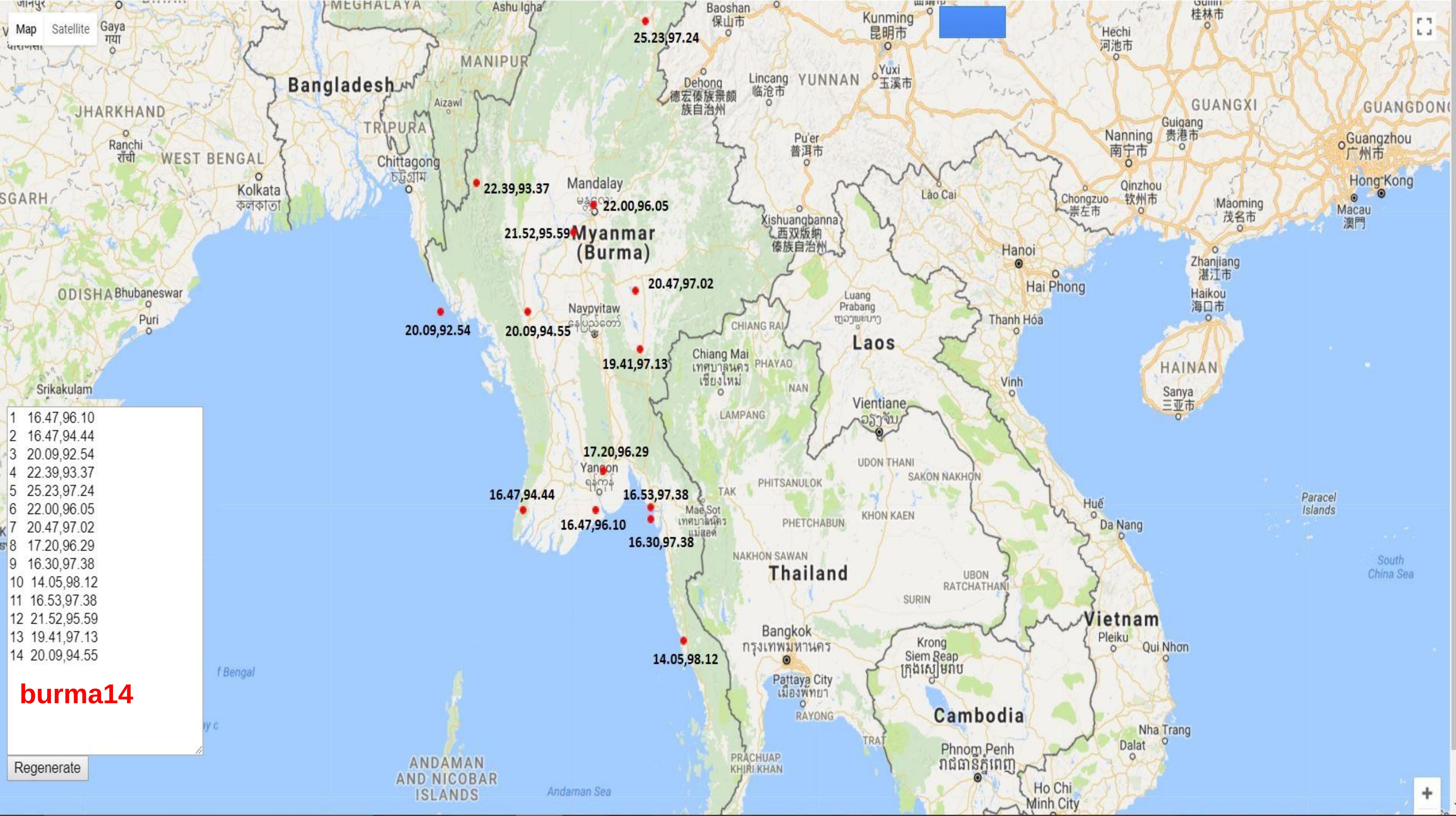


Fig. 4 Illustration of new plants generated by sending 1 long runner from the main plant

Pseudo-code of Discrete PPA

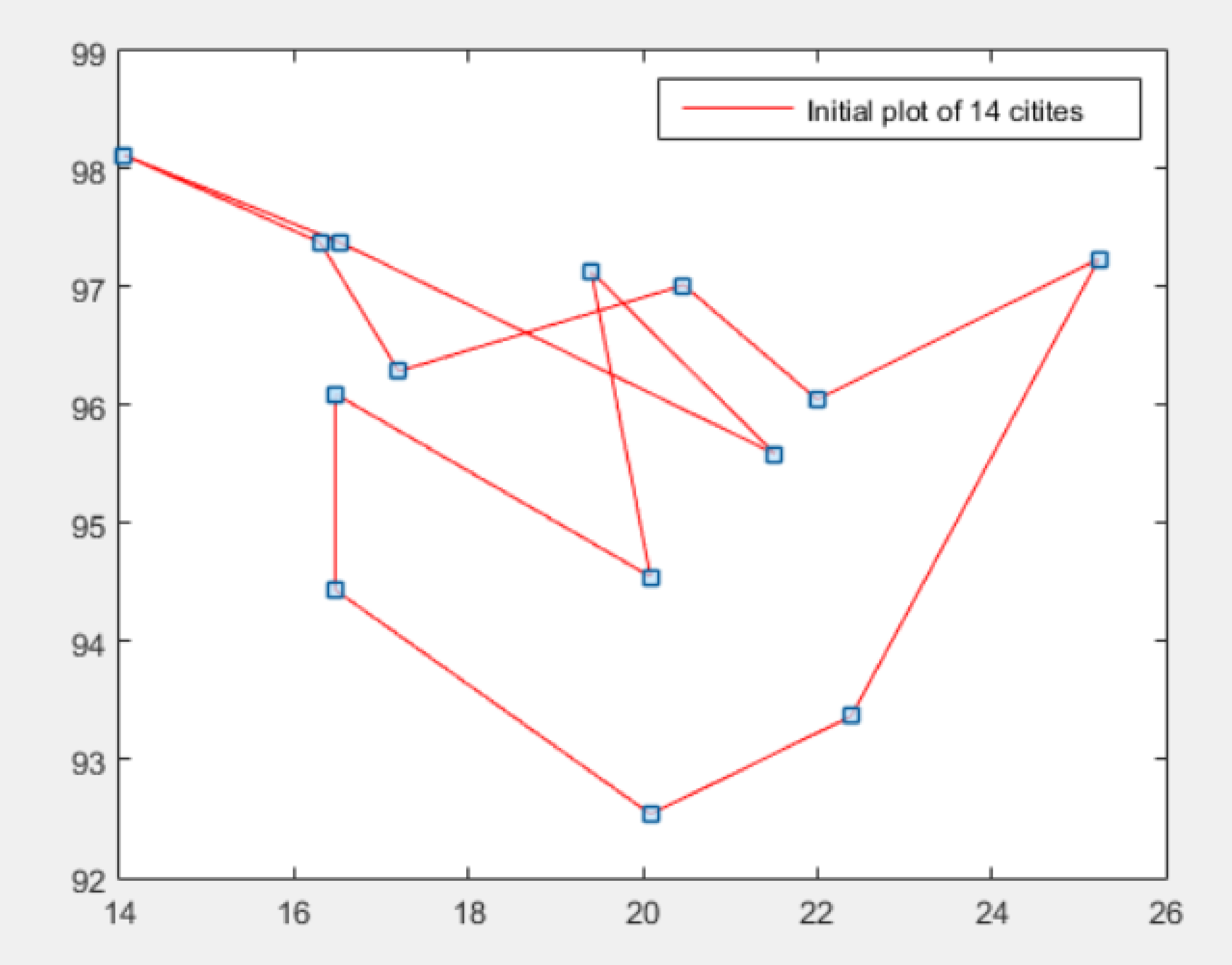
Algorithm 2 Pseudo-code of Discrete PPA

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).



Map
Satellite
1 16.47,96.10
2 16.47,94.44
3 20.09,92.54
4 22.39,93.37
5 25.23,97.24
6 22.00,96.05
7 20.47,97.02
8 17.20,96.29
9 16.30,97.38
10 14.05,98.12
11 16.53,97.38
12 21.52,95.59
13 19.41,97.13
14 20.09,94.55
burma14
Regenerate
25.23,97.24
22.39,93.37
22.00,96.05
21.52,95.59
20.47,97.02
20.09,92.54
20.09,94.55
19.41,97.13
17.20,96.29
16.47,94.44
16.53,97.38
16.47,96.10
16.30,97.38
14.05,98.12
Bangladesh
Myanmar (Burma)
Thailand
Laos
Vietnam
Cambodia
Mandalay
Naypyitaw
Yangon
Bangkok
Hanoi
Kolkata
Chittagong
Vientiane
Chiang Mai
Kunming
Hong Kong
Guangzhou
Phnom Penh
Ho Chi Minh City
Andaman Sea
ANDAMAN AND NICOBAR ISLANDS
HAINAN
South China Sea

1	16.4700	96.1000
2	16.4700	94.4400
3	20.0900	92.5400
4	22.3900	93.3700
5	25.2300	97.2400
6	22	96.0500
7	20.4700	97.0200
8	17.2000	96.2900
9	16.3000	97.3800
10	14.0500	98.1200
11	16.5300	97.3800
12	21.5200	95.5900
13	19.4100	97.1300
14	20.0900	94.5500



Generate a population $P = Xi$, $i = 1, . . . , NP$ of valid tours; NP=50, Total city = 14, (14x50)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
1	1	8	4	2	4	1	11	4	14	12	9	13	5	3	1	2	13	6	7	12	14	6	8	7	8	8	6	3	4	11	12	8	9	12	9	3	13	10	5	13	12	7	6	3	10	12	10	7	14	5
2	2	13	11	7	1	13	12	7	10	13	3	3	14	10	3	4	1	1	11	10	2	9	9	2	3	14	2	9	3	2	7	10	2	13	13	11	2	2	11	8	2	14	5	13	9	14	11	12	5	4
3	9	4	1	3	3	2	2	5	7	8	10	9	3	2	13	8	12	7	6	8	8	5	2	1	2	1	4	4	6	8	5	4	7	4	8	6	4	5	2	14	3	12	13	9	3	1	1	6	7	12
4	5	7	6	4	11	11	3	2	5	5	4	14	11	8	6	1	8	3	5	9	10	13	14	14	9	5	14	6	14	6	4	12	12	2	11	2	5	8	1	4	7	2	14	11	2	11	14	2	1	1
5	3	3	14	1	7	10	1	1	11	10	7	11	8	14	5	6	7	10	8	2	3	14	6	3	10	3	8	7	1	9	11	9	14	1	5	14	14	13	4	11	1	11	2	2	7	8	2	9	8	9
6	10	12	3	6	12	8	7	3	8	9	1	4	6	1	9	7	2	13	3	14	6	7	10	10	7	13	13	12	10	4	13	13	10	14	14	10	1	12	14	9	10	9	4	12	5	7	3	3	3	3
7	14	14	10	14	13	14	6	13	13	7	2	1	7	5	8	14	4	9	14	1	5	11	12	13	13	11	7	11	7	7	9	7	4	9	1	4	9	14	6	6	9	13	12	7	1	10	12	8	2	13
8	11	10	8	9	2	9	8	9	2	6	12	8	4	9	7	3	9	12	13	5	7	8	11	8	1	7	10	10	5	10	2	1	5	7	7	12	6	4	10	1	14	4	7	10	6	3	6	4	4	6
9	7	9	12	8	5	3	9	8	9	4	14	6	1	6	10	10	3	4	12	4	9	4	1	6	14	2	9	13	13	13	10	11	11	11	12	8	3	7	8	3	6	5	10	5	14	6	7	14	13	8
10	13	5	7	5	10	4	10	6	6	3	11	5	10	4	14	9	10	8	4	13	11	12	3	11	11	12	11	5	9	3	1	3	8	3	3	7	11	11	9	10	5	3	3	8	4	5	13	13	6	10
11	6	11	9	10	6	6	5	12	12	1	8	12	12	13	12	11	6	14	9	6	1	1	4	4	6	6	5	8	12	5	14	6	13	10	10	5	12	3	12	2	8	6	11	1	13	4	9	5	10	2
12	4	2	5	11	14	5	13	10	4	2	13	10	9	12	4	13	14	5	10	7	12	3	5	5	5	10	1	1	11	1	3	2	1	6	2	9	8	9	7	12	11	1	9	4	11	13	4	1	12	11
13	12	6	2	13	8	7	4	14	1	14	6	2	13	7	11	12	11	11	1	11	13	10	13	12	4	9	3	14	2	12	6	14	3	5	4	1	10	1	3	5	13	10	8	14	12	2	8	10	11	14
14	8	1	13	12	9	12	14	11	3	11	5	7	2	11	2	5	5	2	2	3	4	2	7	9	12	4	12	2	8	14	8	5	6	8	6	13	7	6	13	7	4	8	1	6	8	9	5	11	9	7

Compute $N_i = f(X_i)$, $\forall X_i \in P$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	0	1.6600	5.0772	6.5191	8.8339	5.5302	4.1044	0.7543	1.2912	3.1523	1.2814	5.0757	3.1152	3.9379
2	1.6600	0	4.0883	6.0159	9.1966	5.7596	4.7599	1.9888	2.9449	4.4044	2.9406	5.1793	3.9849	3.6217
3	5.0772	4.0883	0	2.4452	6.9649	3.9960	4.4961	4.7344	6.1473	8.2230	6.0083	3.3686	4.6401	2.0100
4	6.5191	6.0159	2.4452	0	4.8003	2.7082	4.1242	5.9550	7.2917	9.5978	7.1007	2.3844	4.7977	2.5850
5	8.8339	9.1966	6.9649	4.8003	0	3.4422	4.7651	8.0860	8.9311	11.2146	8.7011	4.0604	5.8210	5.8014
6	5.5302	5.7596	3.9960	2.7082	3.4422	0	1.8116	4.8060	5.8531	8.2151	5.6294	0.6648	2.8062	2.4286
7	4.1044	4.7599	4.4961	4.1242	4.7651	1.8116	0	3.3505	4.1855	6.5136	3.9564	1.7741	1.0657	2.4991
8	0.7543	1.9888	4.7344	5.9550	8.0860	4.8060	3.3505	0	1.4135	3.6430	1.2795	4.3763	2.3643	3.3734
9	1.2912	2.9449	6.1473	7.2917	8.9311	5.8531	4.1855	1.4135	0	2.3686	0.2300	5.5184	3.1200	4.7300
10	3.1523	4.4044	8.2230	9.5978	11.2146	8.2151	6.5136	3.6430	2.3686	0	2.5880	7.8868	5.4507	7.0162
11	1.2814	2.9406	6.0083	7.1007	8.7011	5.6294	3.9564	1.2795	0.2300	2.5880	0	5.3013	2.8908	4.5478
12	5.0757	5.1793	3.3686	2.3844	4.0604	0.6648	1.7741	4.3763	5.5184	7.8868	5.3013	0	2.6122	1.7682
13	3.1152	3.9849	4.6401	4.7977	5.8210	2.8062	1.0657	2.3643	3.1200	5.4507	2.8908	2.6122	0	2.6681
14	3.9379	3.6217	2.0100	2.5850	5.8014	2.4286	2.4991	3.3734	4.7300	7.0162	4.5478	1.7682	2.6681	0

Sort $N = N_i$, $i = 1, . . . , NP$ in ascending order (forminimization);

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
44.9611	45.6573	46.8807	47.8624	48.6215	48.8701	49.5726	49.7025	49.9860	49.9956	50.3201	51.0438	51.9288	52.3027	52.7028	53.6286	54.5348	55.2442	55.2938	55.8028	55.9021	55.9854	56.0224	56.2568	56.4620

26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
57.1946	57.3844	57.5851	58.0682	58.0744	58.5686	58.7461	58.8731	58.9183	59.5009	59.7143	60.3207	61.3942	61.5232	62.1280	62.1664	62.6655	63.0766	63.7991	65.5987	65.7741	65.8467	65.8703	66.2513	72.2481

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
1	2	1	10	14	7	6	8	12	5	7	12	12	4	8	7	7	5	12	3	9	8	3	4	6	9	9	7	2	5	10	8	13	10	12	14	14	6	8	6	3	1	10	9	12	12	7	14	12	10	5
2	4	13	11	2	14	5	3	7	4	11	14	2	3	9	11	12	11	14	9	13	10	13	3	2	13	2	11	4	4	2	3	8	9	7	5	2	2	9	2	13	13	11	2	2	14	14	2	7	2	11
3	8	2	1	8	12	13	2	5	12	6	1	3	6	2	6	6	2	1	4	8	4	9	6	4	8	7	6	8	12	5	2	14	3	5	7	8	4	2	4	9	2	1	7	3	1	12	8	5	5	2
4	1	11	14	10	2	14	9	4	1	5	11	7	14	14	5	2	1	11	6	11	12	11	14	14	11	12	5	1	1	8	9	4	2	4	1	10	14	14	14	11	11	14	12	7	11	2	10	4	8	1
5	6	10	2	3	11	2	10	11	9	8	8	1	1	6	8	9	4	8	7	5	9	2	1	8	5	14	8	6	9	13	10	11	7	11	8	3	8	6	8	2	10	2	14	1	8	11	3	11	13	4
6	7	8	3	6	9	4	7	13	3	3	7	10	10	10	3	3	14	7	12	14	13	12	10	13	14	10	3	7	3	12	7	9	5	13	3	6	13	10	13	12	8	3	10	10	7	9	6	13	12	14
7	14	14	12	5	13	12	13	9	13	14	10	9	7	12	14	8	6	10	11	1	7	7	7	7	1	4	14	14	13	14	13	6	1	9	2	5	7	12	7	7	14	12	4	9	10	13	5	9	14	6
8	3	9	6	7	4	7	1	2	6	13	3	14	5	11	13	4	10	3	10	7	1	10	5	10	7	5	13	3	6	4	1	1	6	2	4	7	10	11	10	10	9	6	5	14	3	4	7	2	4	10
9	10	3	7	9	5	10	14	10	8	12	6	6	13	1	12	14	8	6	13	12	11	5	13	9	12	11	12	10	8	7	14	3	14	10	13	9	9	1	9	5	3	7	11	6	6	5	9	10	7	8
10	9	4	13	11	3	3	11	1	10	4	5	5	9	3	4	13	9	5	5	3	3	8	9	11	3	8	4	9	10	11	11	10	4	1	6	11	11	3	11	8	4	13	8	5	5	3	11	1	11	9
11	11	6	9	1	6	11	6	14	2	9	4	8	12	4	9	5	12	4	8	10	6	1	12	5	10	13	9	11	2	3	6	2	13	14	10	1	5	4	5	1	6	9	13	8	4	6	1	14	3	12
12	13	5	4	12	1	9	5	3	11	10	13	11	11	5	10	1	7	13	1	2	2	4	11	1	2	1	10	13	11	9	5	12	11	3	12	12	1	5	1	4	5	4	1	11	13	1	12	3	9	7
13	12	7	8	13	10	8	4	6	14	1	2	13	2	13	1	10	3	2	14	4	14	14	2	3	4	3	1	12	14	1	4	5	12	6	11	13	3	13	3	14	7	8	3	13	2	10	13	6	1	3
14	5	12	5	4	8	1	12	8	7	2	9	4	8	7	2	11	13	9	2	6	5	6	8	12	6	6	2	5	7	6	12	7	8	8	9	4	12	7	12	6	12	5	6	4	9	8	4	8	6	13

for *i* = 1 : *E(NP/*10*)*, Top 10% of plants **do**

Generate *(y/i)* short runners for plant *i* using 2-opt rule, where y is an arbitrary parameter.
y = 1 (generated arbitrarily) so 1 short runner will be generated for *X1*

2
4
8
1
5
7
14
3
10
9
11
13
12
6

X1

```
swapPosition1 = 5

swapPosition2 = 14
```

2
4
8
1
6
7
14
3
10
9
11
13
12
5

Short runner

Exploit using the short runner

2
4
8
1
5
7
14
3
10
9
11
13
12
6

X1

Path Cost : 44.9611435667493

2
4
8
1
6
7
14
3
10
9
11
13
12
5

Short runner

Path Cost : 47.8227529664884

if $N_i > f(r_i)$ **then**
    $X_i \leftarrow r_i$
else
    Ignore r_i
end if

for $i = E(NP/10) + 1 : NP$ do

ri = 1 runner for plant i using k-opt rule, $k > 2$, 1 long runner for each plant not in the top 10 percent

6
5
13
14
2
4
12
7
10
3
11
9
8
1

X6

So 1 long runner for X6. for *k-opt rule, k>2; k=3,* the 3 swapping positions are

7 2 4

1	6
2	14
3	13
4	12
5	2
6	4
7	5
8	7
9	10
10	3
11	11
12	9
13	8
14	1

Long runner

Explore using the long runner

6
5
13
14
2
4
12
7
10
3
11
9
8
1

X6

Path Cost : 48.8701495669961

1	6
2	14
3	13
4	12
5	2
6	4
7	5
8	7
9	10
10	3
11	11
12	9
13	8
14	1

Long runner

Path Cost : 51.6121726742571

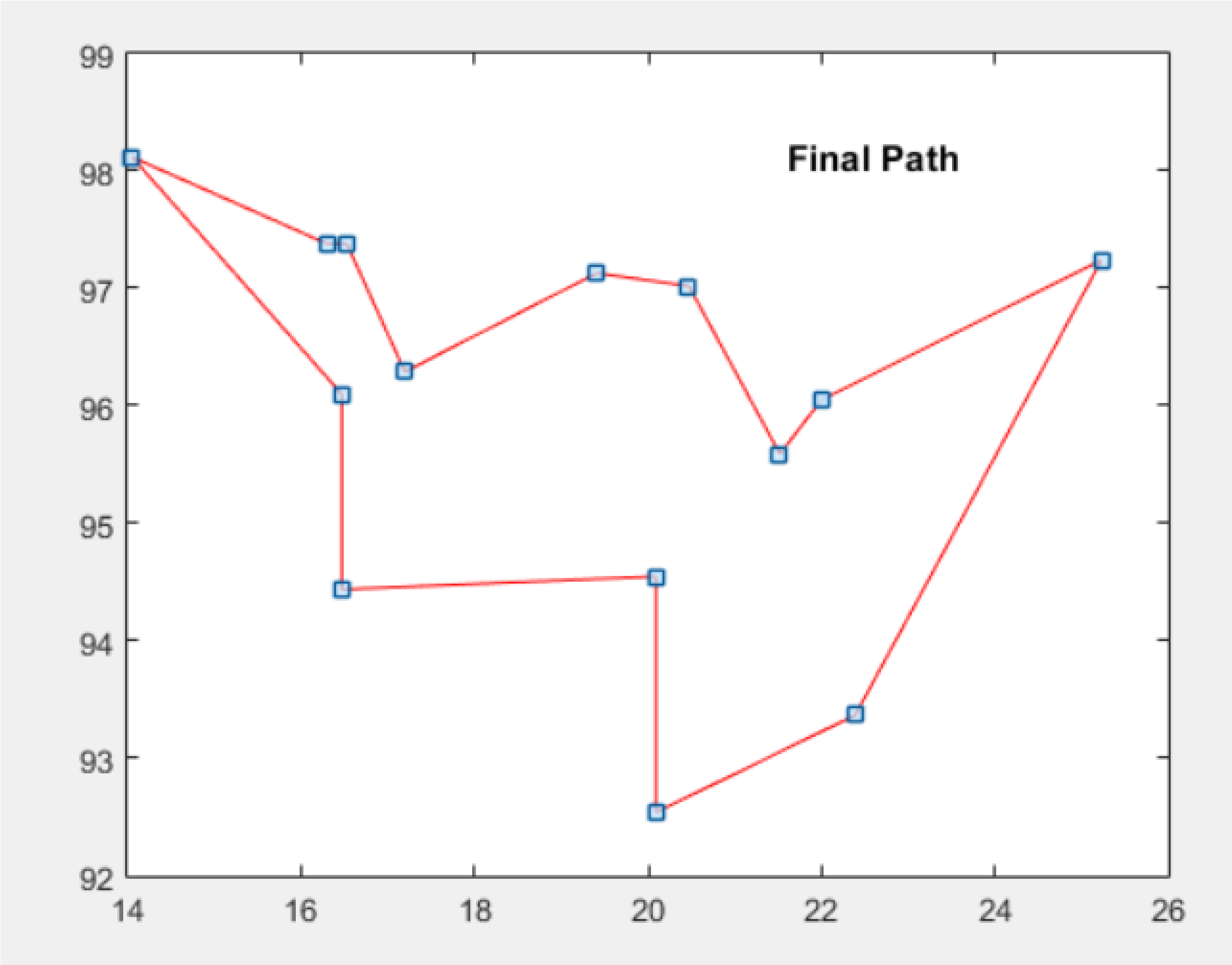
if *Ni > f (ri)* **then**
    Xi ←*ri*
else
    Ignore *ri*
end if

return *P*, (the population of solutions).

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
2	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
3	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
4	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
6	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
7	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
8	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
9	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14
10	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13
11	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7
12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12
13	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
14	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
1	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
2	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9	9
3	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11
4	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	8	1	8
5	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	2	4
6	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	3	3
7	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	4	14
8	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	14	6
9	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	14	12	12
10	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13	13
11	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	8	2
12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	7	1
13	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	7
14	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5

Path Cost : 25.0258364541142



Path: 6 5 4 3 14 2 1 10 9 11 8 13 7 12
Path Cost : 30.8785

References

[1] 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)

[2] https://www.linkedin.com/pulse/strawberry-plant-propagation-dr-tohid-nooralvandi

[3] Sulaiman,M., Salhi, A.: A Seed-based plant propagation algorithm: the feeding station model. Sci World J (2015)

[4] Sulaiman, M., Salhi, A., Fraga, E.S.: The Plant Propagation Algorithm: Modifications and Implementation. ArXiv e-prints (2014)

[5] Sulaiman,M., Salhi, A., Selamoglu, B.I.,Kirikchi, O.B.:Aplant propagation algorithm for constrained engineering optimisation problems. Mathematical Problems in Engineering 627416, 10 pp (2014). doi:10.1155/2014/627416