'Stochastic Optimization'
Maximization of a building surface on a parcel of land.

16 December 2019

—

Heuristic
"Problem of Stochastic Optimization"

—

SINGH Arjun
MOPIDEVI Murali Krishna

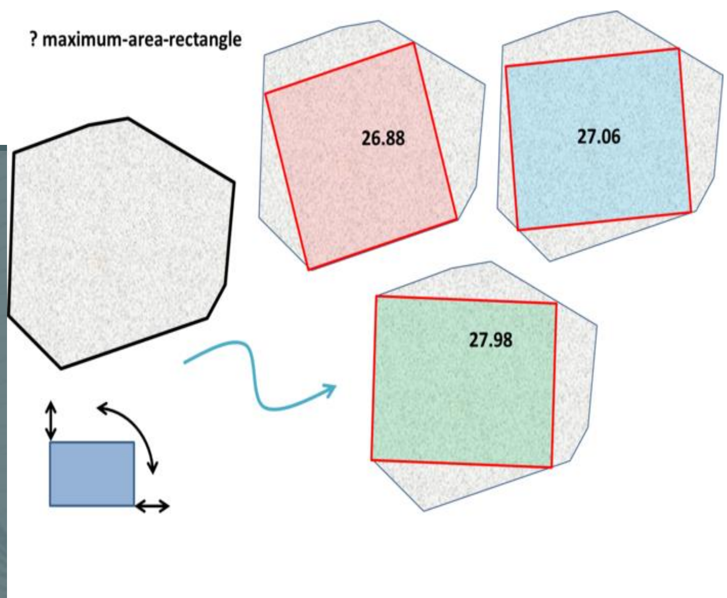
Table of Contents

The Problem

An architectural firm proposes the following simplified problem:

"*find the building with the largest floor area contained in the given parcel*".

It is a Mathematical problem to build an approximation algorithm for finding the largest rectangle inside a non-convex polygon



The Optimization

- The components of the problem are:
    - ⇨ The polygon: the constrained search space;
    - ⇨ The rectangle: the solution of the problem;
    - ⇨ Feasibility (is the rectangle inscribed in the polygon?): A constrained problem;
    - ⇨ The area of the rectangle: the evaluation function;
        - Problem of maximization..
        - In this case we maximize the Area
        - To maximize Area we optimize two points or 4 coordinates and an angle for rotation

The performance of all heuristic algorithms influenced by the search space structure. Consequently, the design of an efficient algorithm needs to exploit, implicitly or explicitly, some features of the search space. For many heuristics, especially local searches, the complexity of the algorithm is very strongly influenced by the asperity of the local structures of local optima

The Process
A Heuristic Approach

The Rectangle

- Limit the number of parameters (reduce the size of the problem);
- "Thinking Neighborhood": be certain that the "neighbor" of a rectangle is a rectangle;
- Choose it's representation in order to efficiently browse the search space

The Search Space

Define the search space of the rectangle according to the polygon :

- Coordinates : bounding box of the polygon;
- Angle : 180° with amplitude ([0; 180], [-90; 90], …)
- We are using two techniques Single Agent and Multi Optimization both have their own way of exploring space
-

Taboo Search

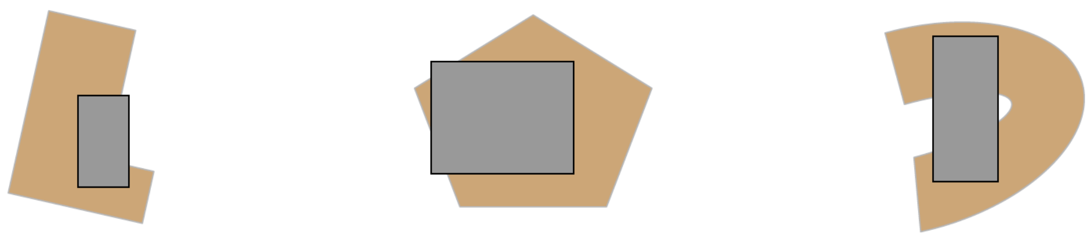
The main objective of the algorithm is to discourage the search process from visiting configurations in some space regions that are already considered as explored.

Particle Swarm Optimization

The PSO algorithm employs a swarm of particles, which traverse a multidimensional search space to seek out optima. Each particle is a potential solution and is influenced by experiences of its neighbors as well as itself

The Feasibility

Validity of the rectangle i.e. does my obtained solution rectangle is maximum that can be obtained in the given polygon



Library called Pyclipper is used to verify that rectangle fits inside the polygon and Clipping algorithm such asVatti, Weiler-Atherton, Greiner-Hormann, Sutherland-Hodgman are used to verify the results

Now the different Polygon shapes available

- polygon = ((10,10),(10,400),(400,400),(400,10))
- polygon = ((10,10),(10,300),(250,300),(350,130),(200,10))
- polygon = ((50,150),(200,50), (350,150),(350,300), (250,300),(200,250), (150,350), (100,250), (100,200))
- polygon = ((50,50),(50,400),(220,310),(220,170),(330,170),(330,480),(450,480),(450,50))

Parameters

Particle Swarm Optimization	Taboo Search
Nb_cycles = 10000 Nb_Indiv = 20 psi, cmax = (0.4, 1.41)	ntaboo = 5 nbNeigh = 40 iterMax = 10000 idemMax = iterMax/10

●Modeling a rectangle as a candidate solution of the problem;

```
def initUn(polygon):
    global xmin,xmax,ymin,ymax
    # xmin = 0
    # xmax = 500
    # ymin = 0
    # ymax = 500
    anglemin = 1.00
    anglemax = 89.00
    boolOK = False
    pos = []
    #print(pos)
    while not boolOK: # as long as it is not feasible
        xo=random.uniform(xmin,xmax)
        yo=random.uniform(ymin,ymax)

        xa=xo+pow(-1,random.randint(0,1))*random.uniform(10,min(xo-xmin,xmax-xo))
        ya=yo+pow(-1,random.randint(0,1))*random.uniform(10,min(yo-ymin,ymax-yo))
        angle = random.uniform(anglemin,anglemax)

        pos = [round(xa),round(ya),round(xo),round(yo),angle]
        rect = pos2rect(pos)
        #print(rect)
        #print('*************')
        #print(pos)
        #print(area(pos2rect(pos)))
        # calcul du clipping
        boolOK = verifyconstraint(rect,polygon)
    #print(pos2rect(pos))
    #print(rect)
```

●Write a function sol2rect(solution) that transforms on solution of the problem into rectangle (n-tuple of coordinates);

```
def pos2rect(pos):
    # coin : point A
    xa, ya = pos[0], pos[1]
    # centre du rectangle : point O
    xo, yo = pos[2], pos[3]
    # angle  AÔD
    angle = pos[4]

    # point D : centre for rotation O, at the Angle alpha
    alpha = pi * angle / 180 # degre en radian
    xd = cos(alpha)*(xa-xo) - sin(alpha)*(ya-yo) + xo
    yd = sin(alpha)*(xa-xo) + cos(alpha)*(ya-yo) + yo
    # point C : symétrique de A, de centre O
    xc, yc = 2*xo - xa, 2*yo - ya
    # point B : symétrique de D, de centre O
    xb, yb = 2*xo - xd, 2*yo - yd

    # round for clipping
    return ((round(xa),round(ya)),(round(xb),round(yb)),(round(xc),round(yc)),(round(xd),round(yd)))


# Distance between 2 points (x1,y1), (x2,y2)
def distance(p1,p2):
    return sqrt((p1[0]-p2[0])**2 + (p1[1]-p2[1])**2)

# Area of Rectangle (A (x1, y1), B (x2, y2), C (x3, y3), D (x4, y4))
# = distance AB * distance BC
def area(pos):
    edge1=(pos[0],pos[1])
    edge2=(pos[1],pos[2])
    return round(distance(edge1[0],edge1[1])* distance(edge2[0],edge2[1]),2)
```

• verifyconstraint(polygon, rectangle) that checks that the rectangle is well contained in the polygon for this already existing algorithm in Pyclipper python package ;

```python
def verifyconstraint(rect, polygon):
    try:
        # Config
        pc = pyclipper.Pyclipper()
        pc.AddPath(polygon, pyclipper.PT_SUBJECT, True)
        pc.AddPath(rect, pyclipper.PT_CLIP, True)
        # Clipping
        clip = pc.Execute(pyclipper.CT_INTERSECTION, pyclipper.PFT_EVENODD, pyclipper.PFT_EVENODD)
        #all(iterable) return True if all elements of the iterable are true (or if the iterable is empty)
        return (clip!=[]) and (len(clip[0])==len(rect)) and all(list(map(lambda e:list(e) in clip[0], rect)))
    except pyclipper.ClipperException:
        # print rect
        return False
```

Particle Swarm Optimization :The Move

```python
# Calculate the velocity and move a paticule
def move(particle,dim):
    global c1,c2,psi,cmax
    nv = dict(particle)
    #everytime a new dictionary is created for the single particle of swarm
    #print('------------------*****************-----------')
    #print(nv)
    #break;
    #velocity for 5 parameters
    velocity = [0]*dim
    for i in range(dim):
        velocity[i] = (particle["vit"][i]*psi + \
        cmax*random.uniform()*(particle["bestpos"][i] - particle["pos"][i]) + \
        cmax*random.uniform()*(particle["bestvois"][i] - particle["pos"][i]))
    position = [0]*dim
    for i in range(dim):
        position[i] = particle["pos"][i] + velocity[i]
    # New position might contradict my dimension constraint
    # i need to verify that cordinates and angle give me a rectangle
    # which lies in the polgon plot of land
    #print("the angle is ")
    #print(position[4])
    if (position[4] <1.00 or position[4]>89.99):
        position = particle["pos"]
        #move(particle,dim)
    if (verifyconstraint(pos2rect(position),polygon) == False):
        position = particle["pos"]
    #breakpoint()
    nv['vit'] = velocity
    nv['pos'] = position
    nv['area'] = round(area(pos2rect(position)),2)
    return nv
```

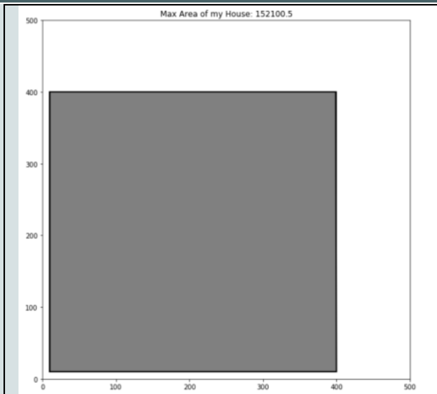
Taboo Search : The best Neighbor

```python
def bestNeighbor(nbNeigh, ltaboo):
    global bestV, bestDist
    #list of indices to swap to generate Neighbors
    lperm = [initdeux(polygon) for a in range(nbNeigh)]
    #print(' --------------- The neighbours are as follows -----------')
    #print(lperm)
    #print('---------------------------------------------------------')
    #breakpoint()
    # case of the first neighbor
    prem = lperm.pop(0)
    best_neighbor = pos2rect(prem['pos'])
    best_n_area = area(pos2rect(prem['pos']))
    #breakpoint()
    for i in range(nbNeigh-1):
        Neigh = pos2rect(lperm[i]['pos'])
        if Neigh not in ltaboo:
                d = area(Neigh)
                if (d > best_n_area):
                    best_neighbor = Neigh
                    best_n_area = d
    return (best_neighbor,best_n_area)
```

Results

Particle Swarm Optimization*

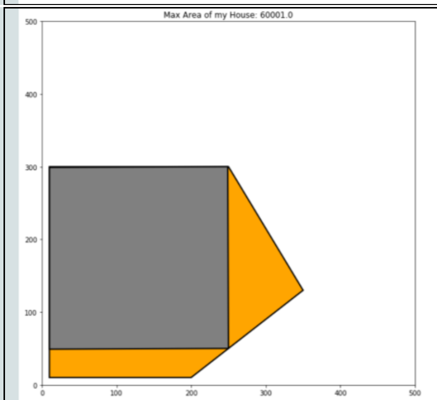
Polygon 1




```
Best of Best
Mean is : 148654.21
Standard deviation : 9926.28
```

```
Mean Accuracy : 97.73
```

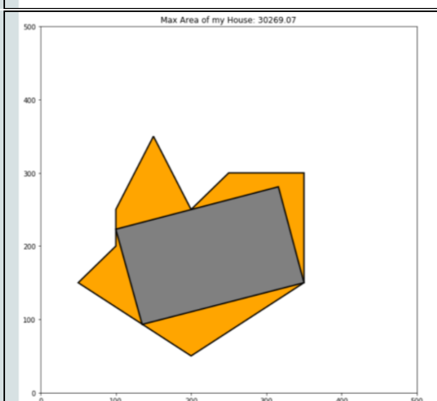
Polygon 2




```
Best of Best
Mean is : 55956.97
Standard deviation : 3927.05
```

```
Mean Accuracy : 93.26666666666667
```

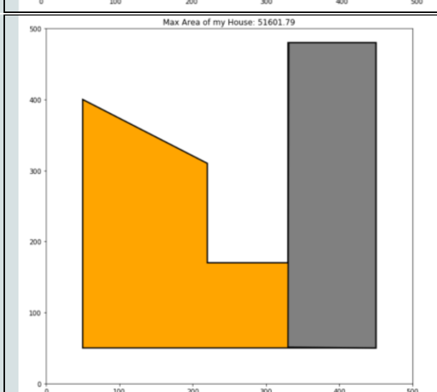
Polygon 3




```
Best of Best
Mean is : 28848.7
Standard deviation : 1923.05
```

```
Mean Accuracy : 95.30333333333333
```

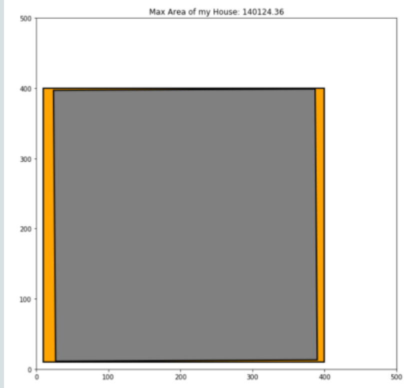
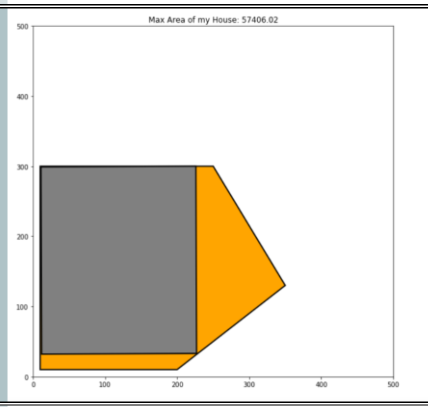
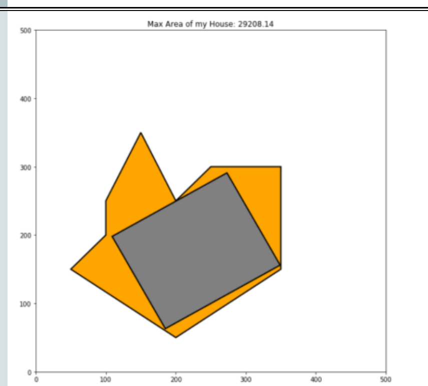
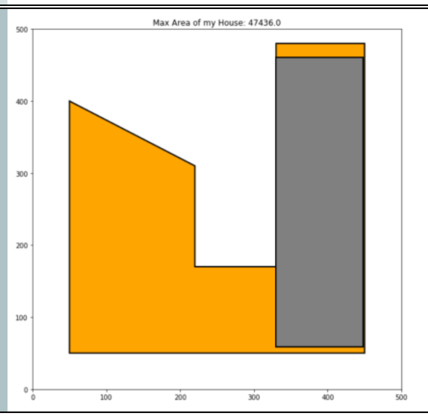
Polygon 4




```
Best of Best
Mean is : 46419.13
Standard deviation : 5677.72
```

```
Mean Accuracy : 89.96333333333334
```

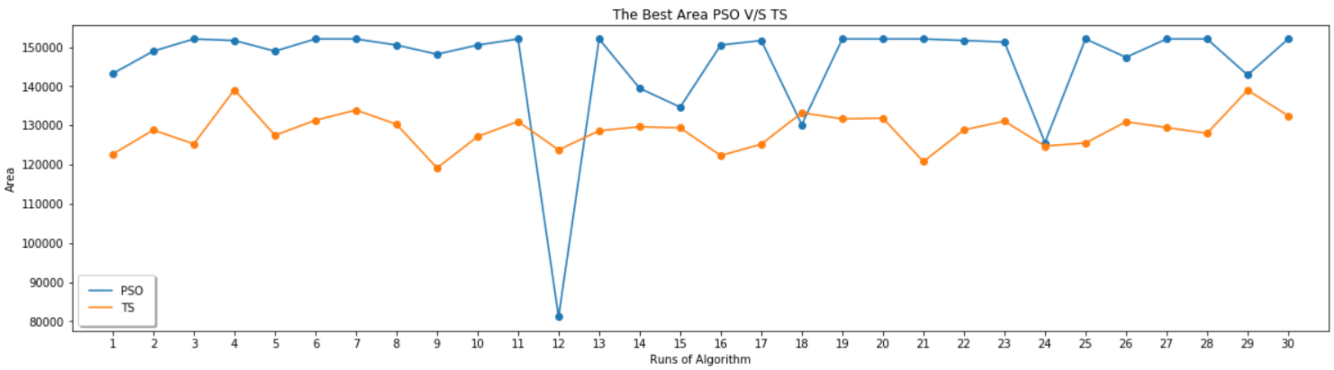
Taboo Search*

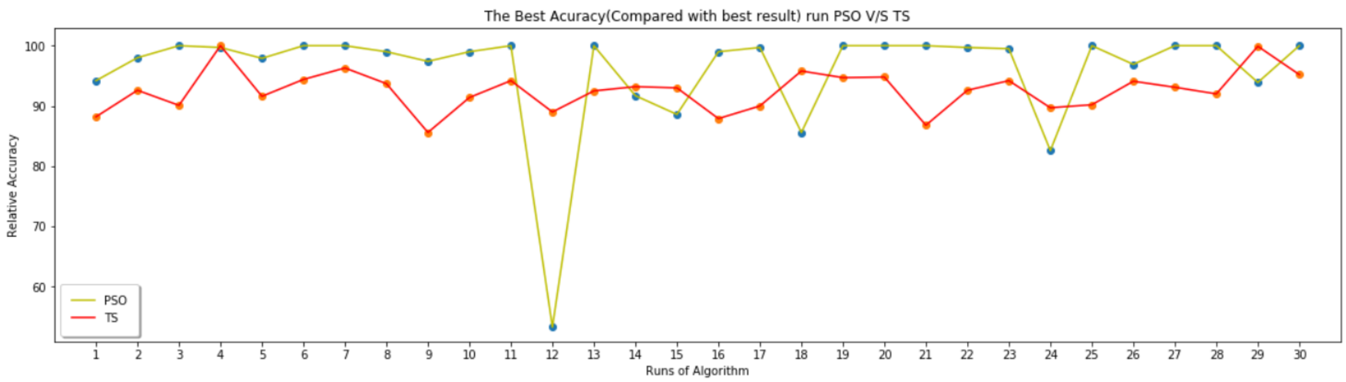
The Polygon	The Best Polygon (Sample 30)	
Polygon 1		Best of Best Mean is : 129048.79 Standard deviation : 5254.42 Accuracy : 92.08333333333333
Polygon 2		Best of Best Mean is : 53646.02 Standard deviation : 1406.83 Accuracy : 93.45666666666666
Polygon 3		Best of Best Mean is : 28226.55 Standard deviation : 626.92 Accuracy : 96.64666666666666
Polygon 4		Best of Best Mean is : 42702.22 Standard deviation : 2033.43 Accuracy : 90.02333333333334

*Models Accuracy Measure is the accuracy of one run/Mean Run obtained in no way it is the actual accuracy. The word accuracy is just the terminology used to describe how frequent the algorithm found the result in close proximity of the average of 30 samples

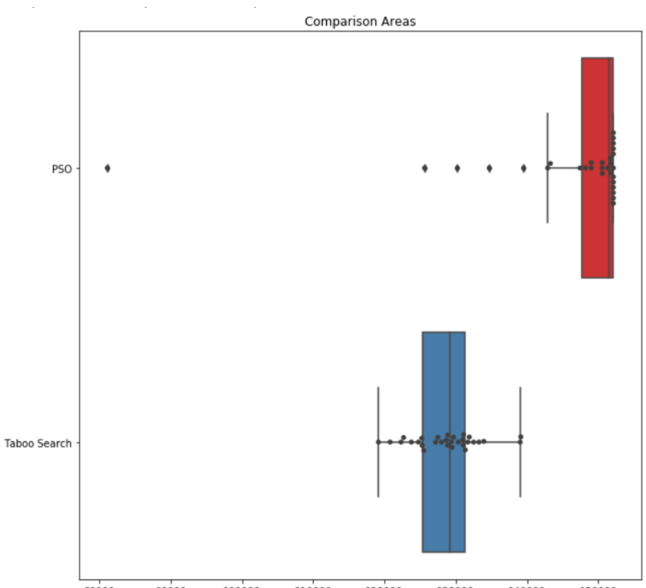
PSO V/S Taboo Search

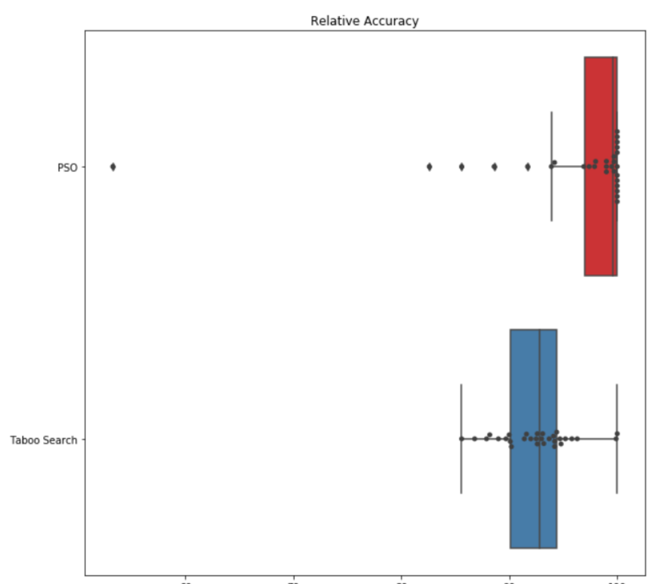
Polygon1: An iteration-by-iteration comparison of Algorithm



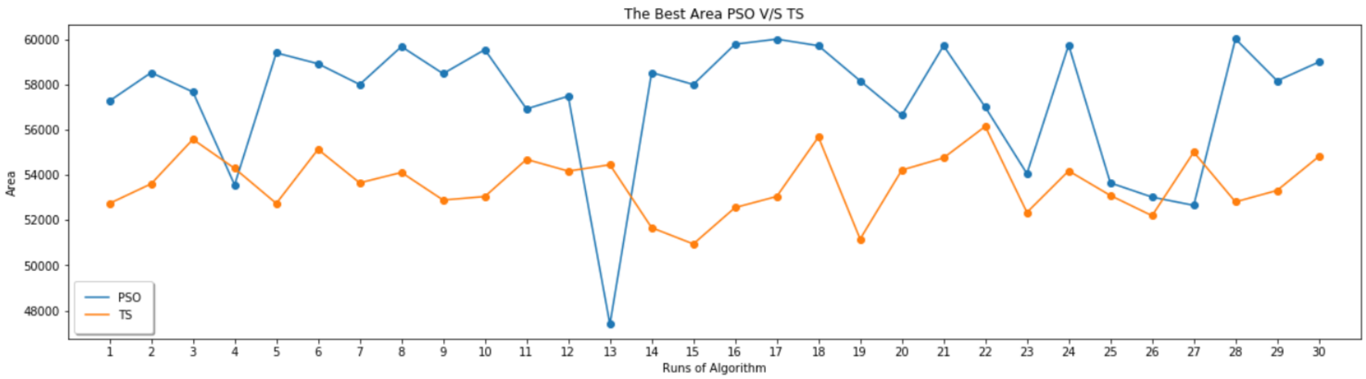


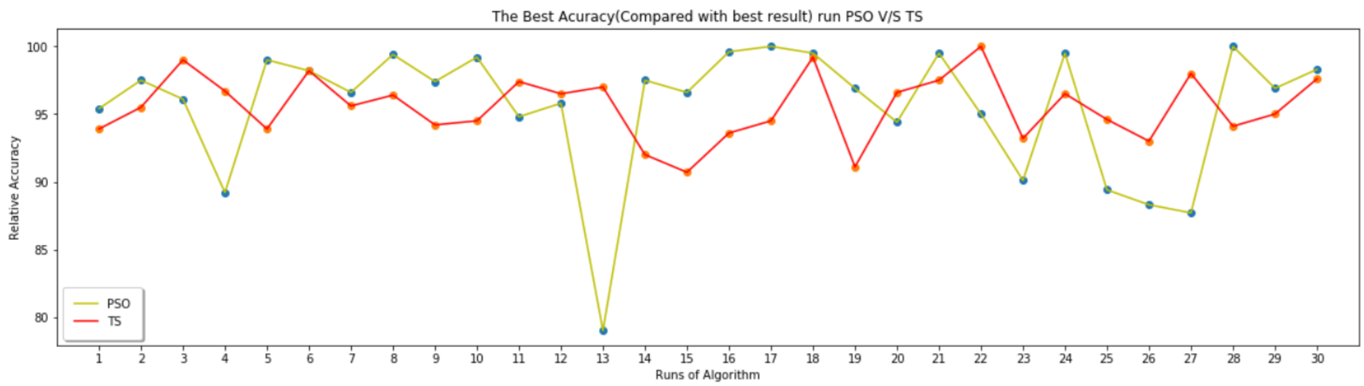
Bar Plot(& Swarm Plot)



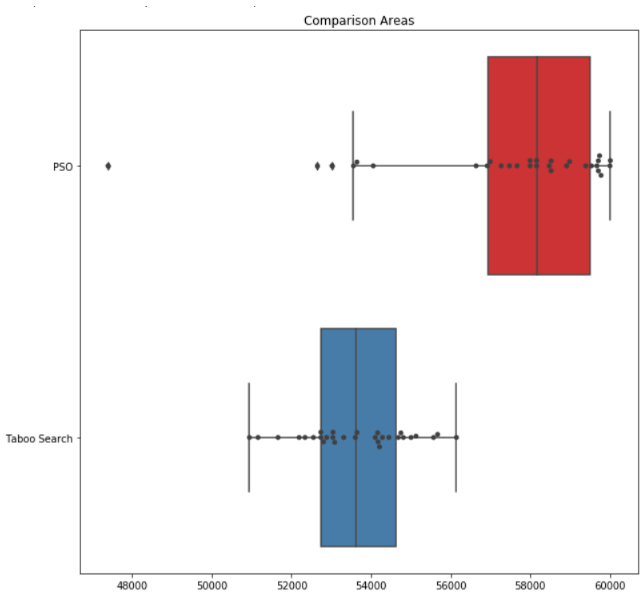


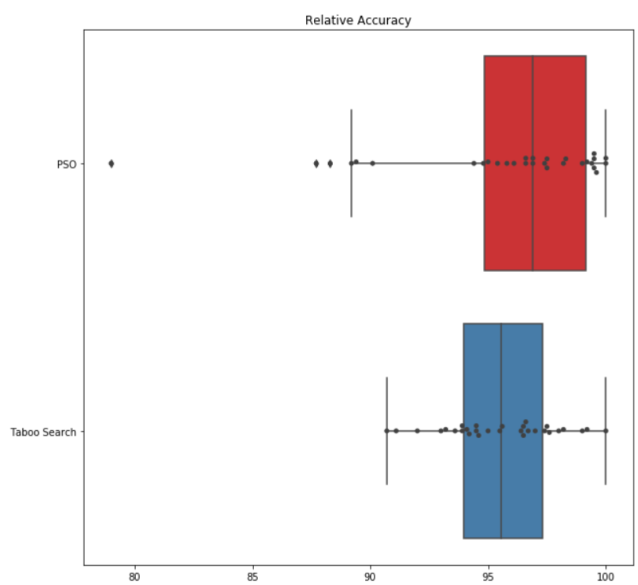
Polygon2: An iteration-by-iteration comparison of Algorithm



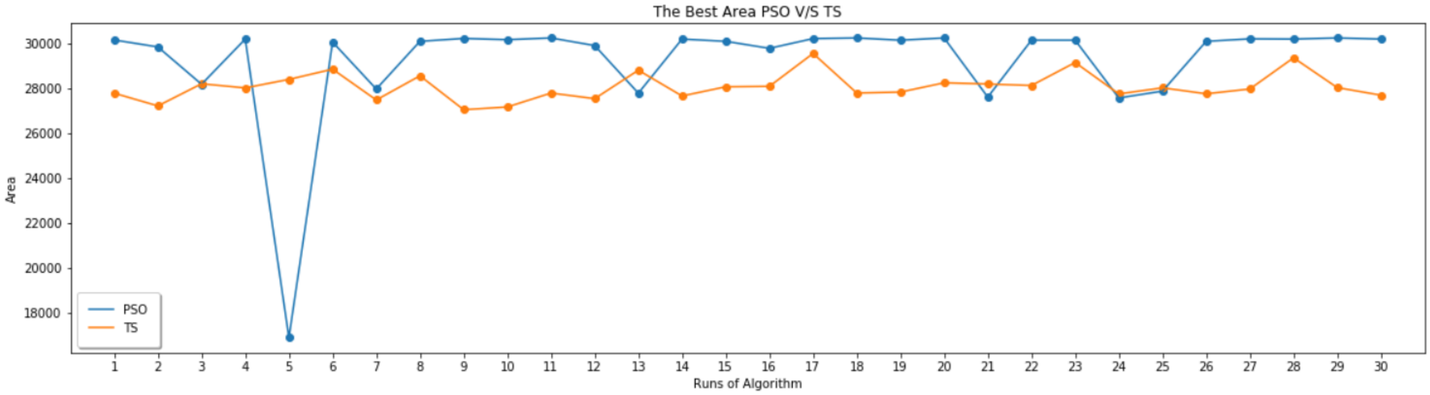


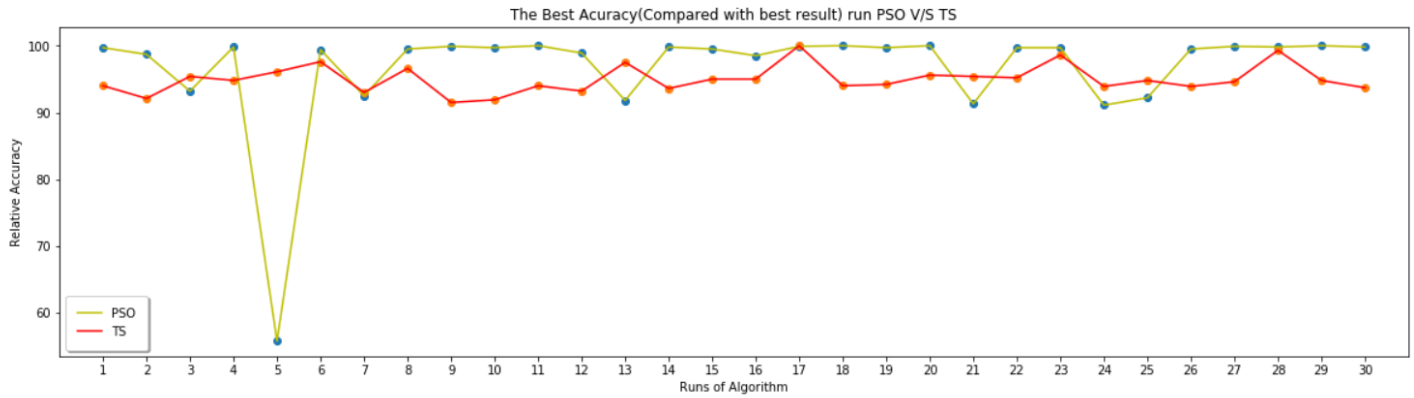
Bar Plot(& Swarm Plot)



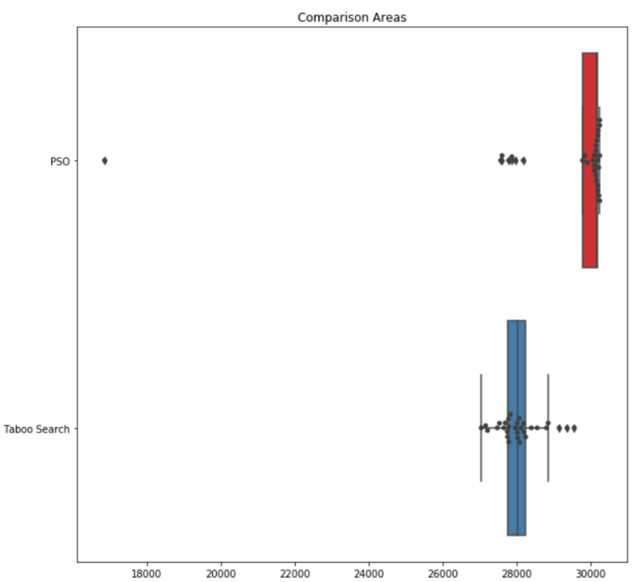


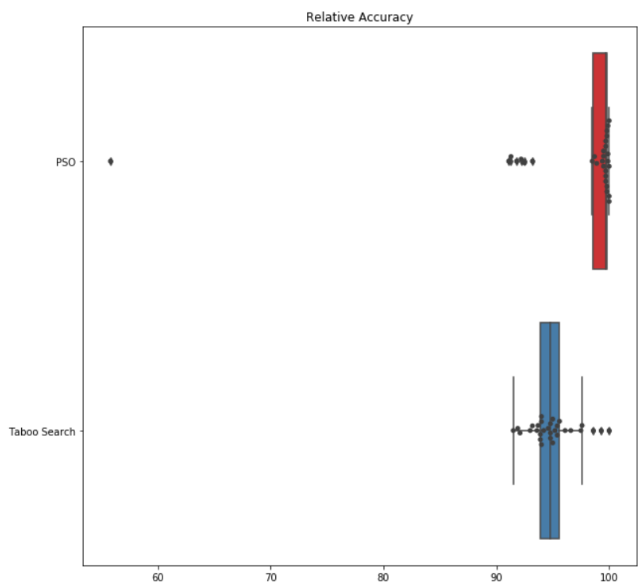
Polygon3: An iteration-by-iteration comparison of Algorithm



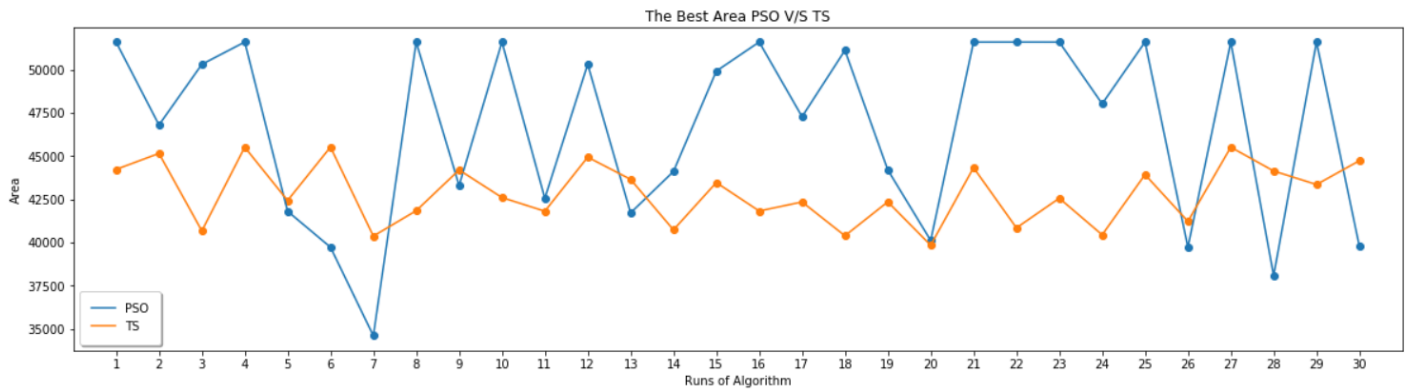


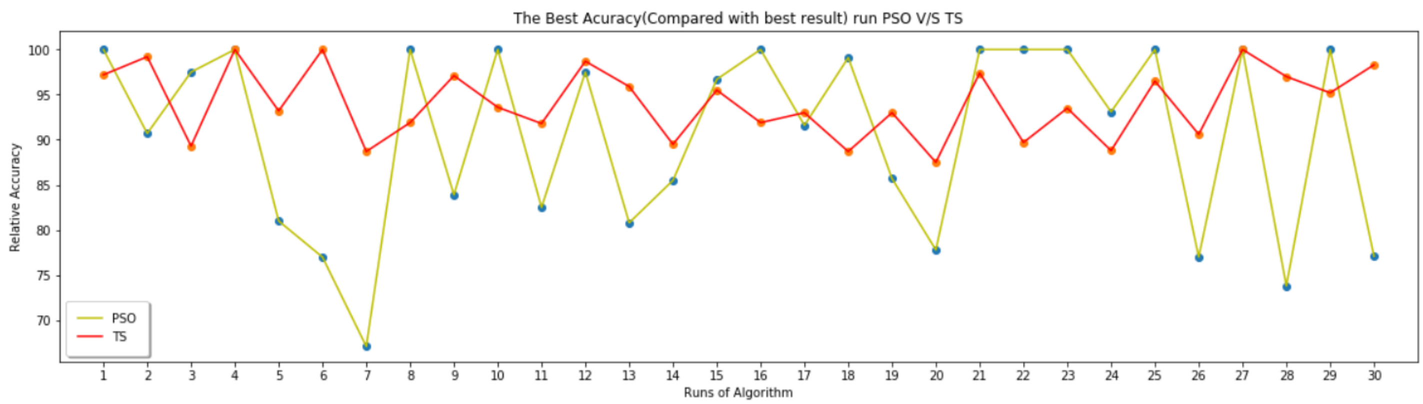
Bar Plot(& Swarm Plot)



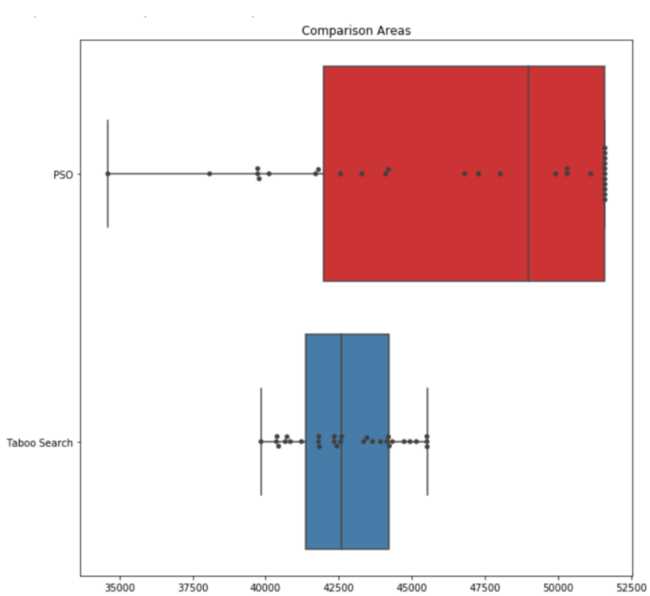


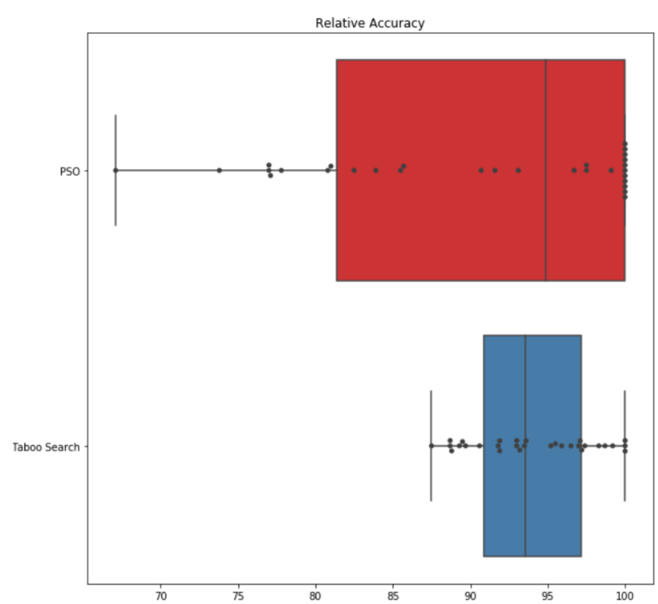
Polygon4: An iteration-by-iteration comparison of Algorithm





Bar Plot(& Swarm Plot)





Testing the Comparison

To test the best Algorithm I applied t-Test for two different means to see which one has best mean, the best result. I obtained following result for polygon 1(square plot) at 95% accuracy the PSO is ahead of Taboo Search by a significant factor

Polygon 1

```
[19] a = sqrt((var(final1['Best Area']) +  var(final['Best Area']))/30)
```

```
[20] b = mean(final1['Best Area']) - mean(final['Best Area'])
```

```
#alpha = 95% confidence
Z_alpha = 1.96
if (b/a< Z_alpha):
  print('mean of TABU Search is best')
  print(b/a)
if (b/a> Z_alpha):
  print('mean of PSO is best')
  print(b/a)
if (b/a == Z_alpha):
  print('mean of the two algorithms is same')
  print(b/a)
```

```
mean of PSO is best
10.629011196189738
```

Polygon 2

*Using t-Test for two different sample of means *

```
[38] a = sqrt((var(final1['Best Area']) +  var(final['Best Area']))/30)
```

```
[39] b = mean(final1['Best Area']) - mean(final['Best Area'])
```

```
[40]
#alpha = 95% confidence
Z_alpha = 1.96
if (b/a< Z_alpha):
  print('mean of TABU Search is best')
  print(b/a)
if (b/a> Z_alpha):
  print('mean of PSO is best')
  print(b/a)
if (b/a == Z_alpha):
  print('mean of the two algorithms is same')
  print(b/a)
```

```
mean of PSO is best
4.290756884395652
```

Polygon 3

*Using t-Test for two different sample of means *

```
[ ] a = sqrt((var(final1['Best Area']) +  var(final['Best Area']))/30)
```

```
[ ] b = mean(final1['Best Area']) - mean(final['Best Area'])
```

```
[ ]
#alpha = 95% confidence
Z_alpha = 1.96
if (b/a< Z_alpha):
  print('mean of TABU Search is best')
  print(b/a)
if (b/a> Z_alpha):
  print('mean of PSO is best')
  print(b/a)
if (b/a == Z_alpha):
  print('mean of the two algorithms is same')
  print(b/a)
```

```
mean of PSO is best
5.528656328269246
```

Polygon 4

*Using t-Test for two different sample of means *

```
[17] a = sqrt((var(final1['Best Area']) +  var(final['Best Area']))/30)
```

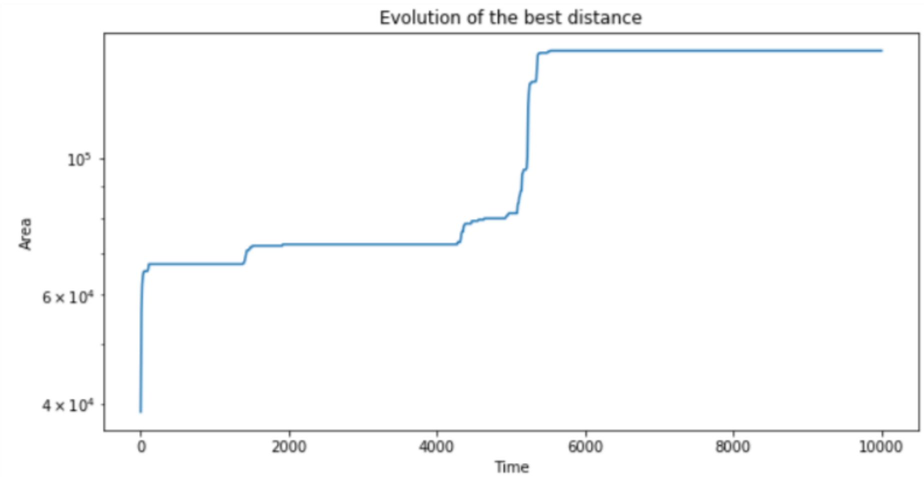
```
[18] b = mean(final1['Best Area']) - mean(final['Best Area'])
```

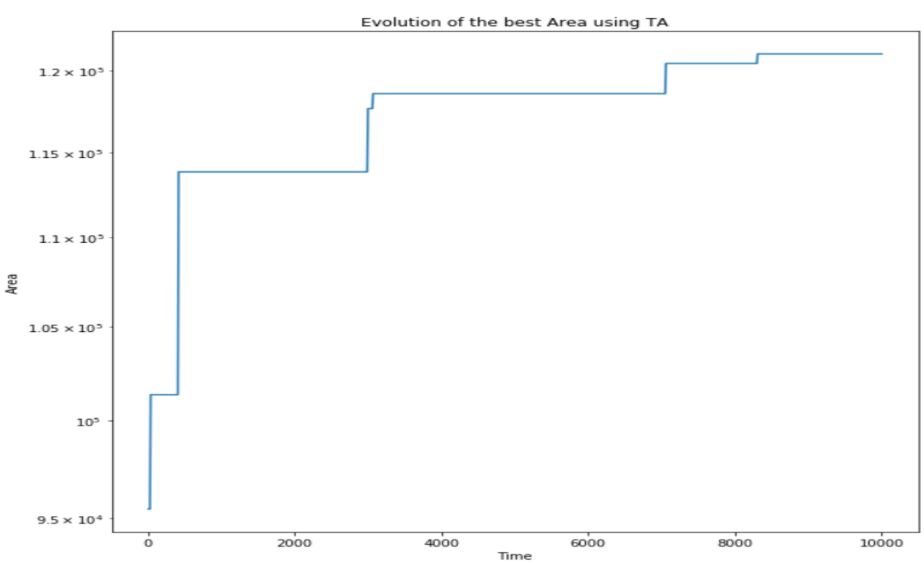
```
[19]
#alpha = 95% confidence
Z_alpha = 1.96
if (b/a< Z_alpha):
  print('mean of TABU Search is best')
  print(b/a)
if (b/a> Z_alpha):
  print('mean of PSO is best')
  print(b/a)
if (b/a == Z_alpha):
  print('mean of the two algorithms is same')
  print(b/a)
```

```
mean of PSO is best
2.269713461804123
```

Conclusion

- The performance of algorithm is very much clear





PSO converges Faster as compared to Taboo Search because it's a multi agent search optimization and having maximized solution most of the time….PSO has a probability of 0.2 to arrive at the best area and a high percentage of optimal solutions obtained are very close to best(see the Box plots)

- Taboo Search on other hand gives a very consistent result with no spikes and least(or no) outliers as shown in box plot as such the consistency of algorithm is good to test and we can further optimize the algorithm to enhance our results
- For the moment Real estate development could really benefit from the use of PSO to find the max and best fit rectangular house on available plot of land

Improvements

- Improvements to algorithm will happen when we give a more guided exploration to both my algorithm. Since they are stochastic they arrive at local minimization, that is why such contradicting results
- For Taboo search when exploring the neighbor, we need to control that neighbor search and optimize the memory usage (searching the taboo list, best is quick sort to look for faster answers from it) of the algorithm to arrive at best solution.