"P7" Búsqueda Local

NESTOR

Marzo 2022

1. Objetivo

El objetivo de la práctica consiste en una variante unidimensional, a partir de un punto seleccionado al azar. Los movimientos locales, por medio de una búsqueda local aleatoria con restricciones en los ejes. Se realizarán réplicas, y el menor de ellos es el resultado de la búsqueda en sí. [1]

2. Desarrollo

Basando el desarrollo en la codificación implementado por E. Schaeffer [1] y se encuentra todas las instrucciones en el repositorio repositorio de Nestor en GitHub.

Para comenzar se hace primero generar la función para graficar con los límites y los pasos. implementados por E. Schaeffer, como se muestra a continuación en el código.

Código 1: Generamos la función

```
import numpy as np
import math as ma
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
def g(x, y):
 return (1 - x/2 + x**5 + y**3) * np.exp(-x**2 - y**2)
 low = -2.7
 high = -low
 step = 0.10
```

Posteriormente generaramos cuantos son los que queremos de los puntos ó muestras con los ciclos ó pasos. Para esta práctica se usará 5 puntos ó muestras con 500 ciclos ó pasos con las posiciones. al tener las celdas se crea generar el cuadro con las funciones que es como inicia el programa como vemos en el código 2.

Código 2: Generamos los puntos ó muestras y los ciclos ó pasos

```
puntos= 5
ciclos= 500
posx = [uniform(low, high) for s in range(puntos)]
posy = [uniform(low, high)for s in range(puntos)]
bestx = posx
besty = posy
```

Se realiza un ciclo for con la funcion movimientos para las posiciones, se muestra a continuación en el código 3. Ya que esto nos muestra que en cada punto va a revisar sus vecinos y va a tomar la mejor posición, por cada punto revisa la posición y revisa sus vecinos y si hay un vecino más grande de mejor posición ese será la nueva posición.

Código 3: Ejecución de códigos para los vecinos y posiciones

```
def vecinos(vecindad):
    pos=[]
    for a in vecindad:
        pos.append((g(a[0],a[1]),a[0],a[1]))
    return(pos)

def movimientos(x, y, bx, by):
  pasos=[]
  listax,listay=[],[]
  listamxy=[]
```

```
    for s in range(puntos):
    mejorx,mejory=bx[0],by[0]
    cx, cy= x[s], y[s]
    Dx=uniform(0, step/3)
    Dy=uniform(0, step/3)
    xi, xd= (cx-Dx), (cx+Dx)
    yi, yd= (cy-Dy), (cy+Dy)
    xi = low if xi < low else xi
    xd = high if xd > high else xd
    yi = low if yi < low else yi
    yd = high if yd > high else yd
    vecindad=[(xi,yd),(cx,yd),(xd,yd),(xi,cy),(xd,cy),(xi,yi),(cx,yi),(xd,yi)]
    datos=vecinos(vecindad)
    pos,cx,cy= (max(datos))
    if pos > g(mejorx,mejory):
    mejorx, mejory=cx,cy
    else:
    mejorx, mejory= bx[0], by[0]
```

3. Resultados

El eje horizontal representa los valores de las x, mientras que el eje vertical representa los valores de las y. Cada uno de los puntos rojos que se observan es un agente que realiza un movimiento, Δx y Δy cuyas combinaciones se hacen las posiciones.

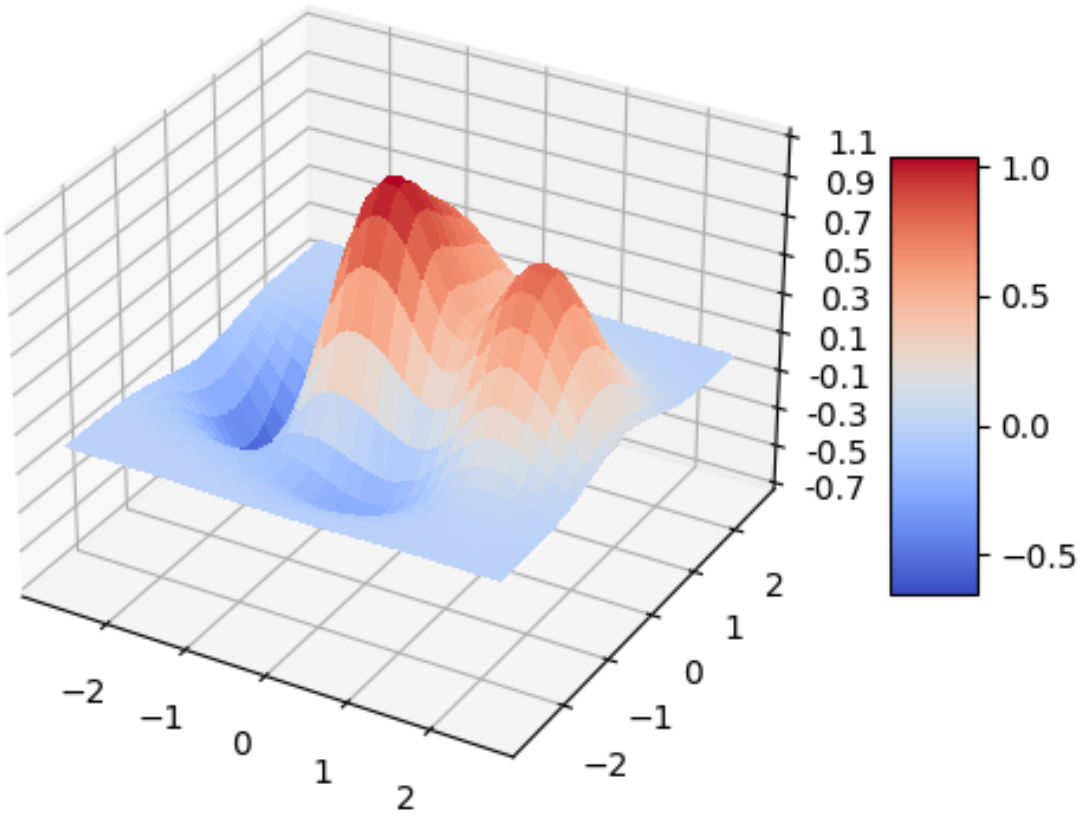


Figura 1: Diagrama en tres dimensiones.

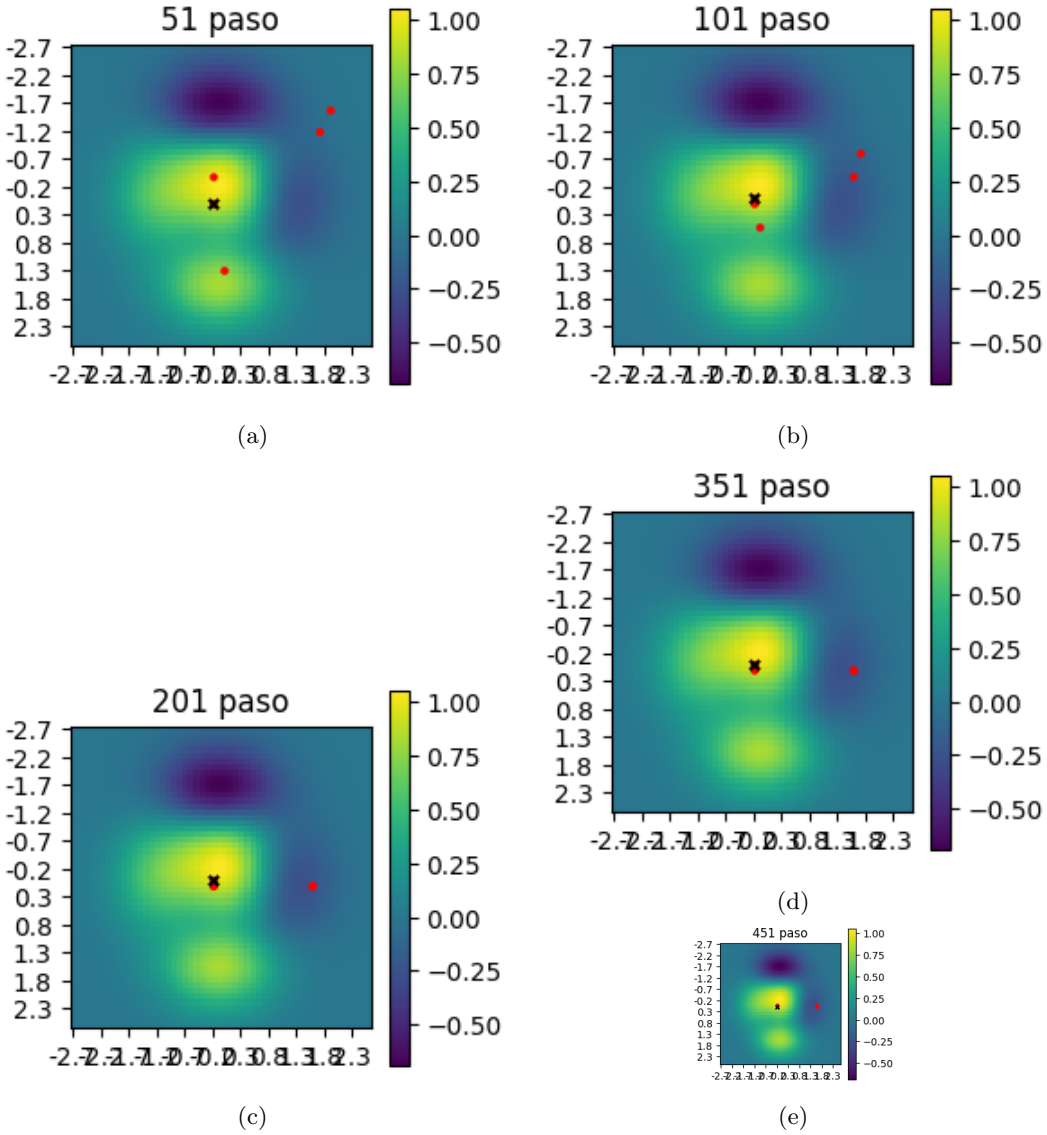


Figura 2: Intervalos de posiciones

4. Conclusiones

Se concluye que los valores máximos y mínimos de una función con los movimientos aleatorios esto nos da la posición de los puntos que se evalúa en los vecinos para comparar si es mejor para ser la nueva posición.

5. Reto 1 Cambio de regla del movimiento

En este reto consiste en cambiar la regla del movimiento de una solución en X (un vector de dimensión arbitraria) a la siguiente a la de recocido simulado: para optimizar una función [1]. Aquí T es una temperatura que decrece en aquellos pasos donde se acepta una empeora; la reducción se logra multiplicando el valor actual de T. A continuación se muestra en las siguientes instrucciones:

Código 4: Temperatura

```
temperatura=100
##temperatura = [100 for i in range(puntos)]
for img in range(ciclos):
    ps,posx,posy,best,temperatura = movimientos(posx, posy, bestx,besty, temperatura)
```

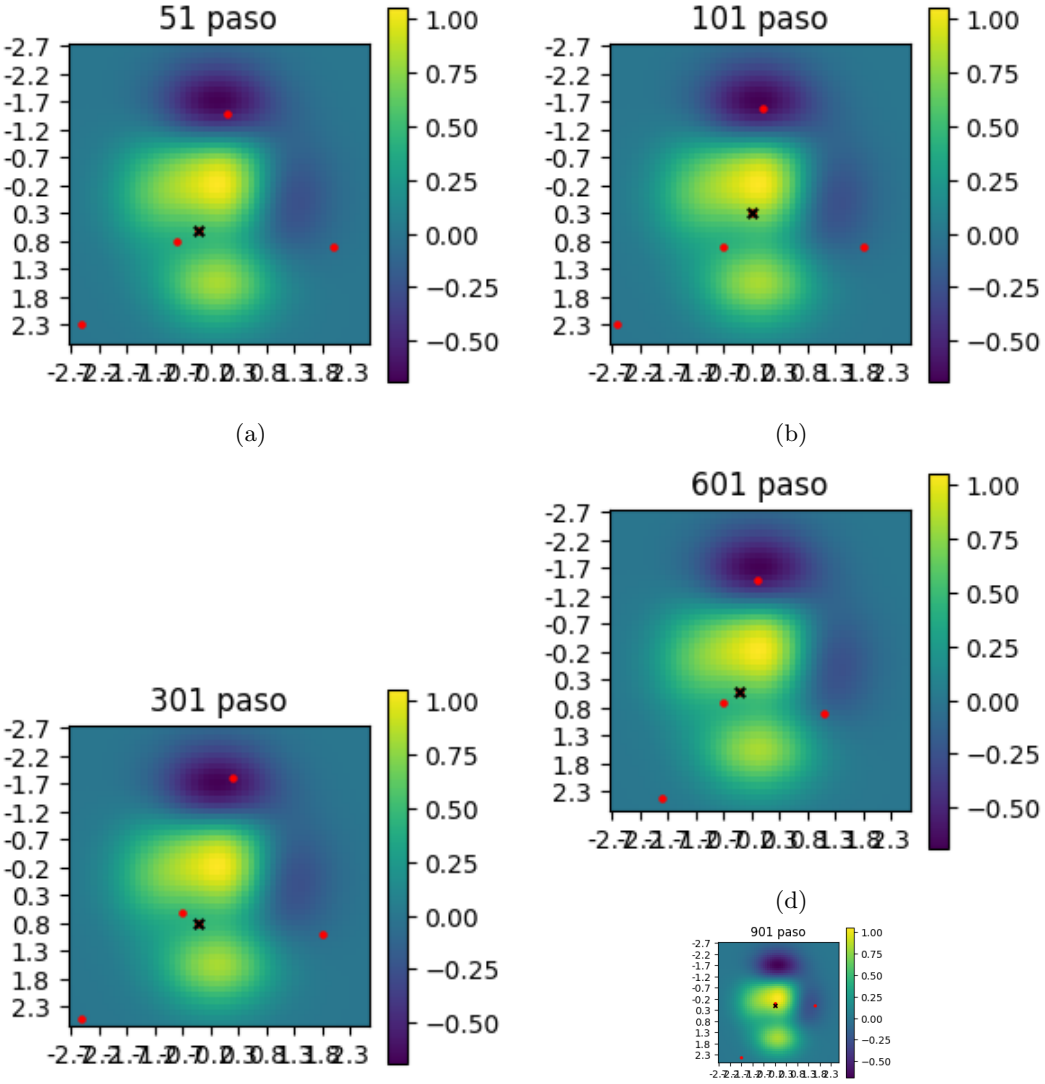
Para este reto la vecindad se vuelve a tomar las mismas decisiones, en cada punto inicial que se revisa su vecindad, cada vecindad se selecciona al azar y se remueve en la lista para que el siguiente ciclo no lo toque.

Código 5: Vecindad

```
vecindad=[(xi,yd),(cx,yd),(xd,yd),(xi,cy),(xd,cy),(xi,yi),(cx,yi),(xd,yi)]
  for k in range(len(vecindad)):
    xnew,ynew = choice(vecindad)
    vecindad.remove((xnew,ynew))
    delta= g(xnew,ynew) - g(cx,cy)
    probabilidad= exp(delta/temp)
    if delta > 0:
    cx, cy=xnew, ynew
    break
    else:
    if random() < (probabilidad):
    cx, cy=xnew, ynew
    temp=(temp*(0.995))
    break
    else:
        cx, cy = cx, cy
```

6. Reto 1 Resultados

Esto se visualiza el comportamiento de la variación respecto a la temperatura. Se encuentra todos los resultados y gifs en el repositorio repositorio de Nestor en GitHub.



(a) (b)

(c) (d)

(e)

Figura 3: Intervalos de Temperatura

7. Reto 2 Comparación de diferencia estadísticamente

En este reto consiste en comparar sí o no hay diferencia estadísticamente significativa entre el método de la tarea base y el método del primer reto en términos de la precisión del resultado obtenido (es decir, la diferencia entre el resultado reportado y el óptimo global) en función del número de iteraciones y el número de réplicas [1]. A continuación se muestra en las siguientes instrucciones:

Código 6: Comparación de iteraciones

```
iteraciones= (100,500,1000)
CB_est2=[]
for f in iteraciones:
    estimados2=[]
    for r in range(replicas):
        posx = [uniform(low, high) for s in range(puntos)]
        posy = [uniform(low, high)for s in range(puntos)]
        bestx = posx
        besty = posy
        temperatura=1000
        for img in range(f):
            ps,posx,posy,best,temperatura = mov_reto1(posx, posy, bestx,besty, temperatura)
            if img==(f-1):
                mejor=max(best)
                estimados2.append(mejor[0])
    CB_est2.append(estimados2)
plt.boxplot([CB_est2[0], CB_est2[1], CB_est2[2]])
plt.xlabel('iteraciones')
plt.ylabel('Posiciones finales maximizadas')
plt.xticks([1,2,3], ['100','500','1000'])
```

8. Reto 2 Resultados

Esto se visualiza la diferencia entre los resultado reportado y la función del número de iteraciones y el número de réplicas estadísticamente. Se encuentra todos los resultados y gifs en el repositorio repositorio de Nestor en GitHub.

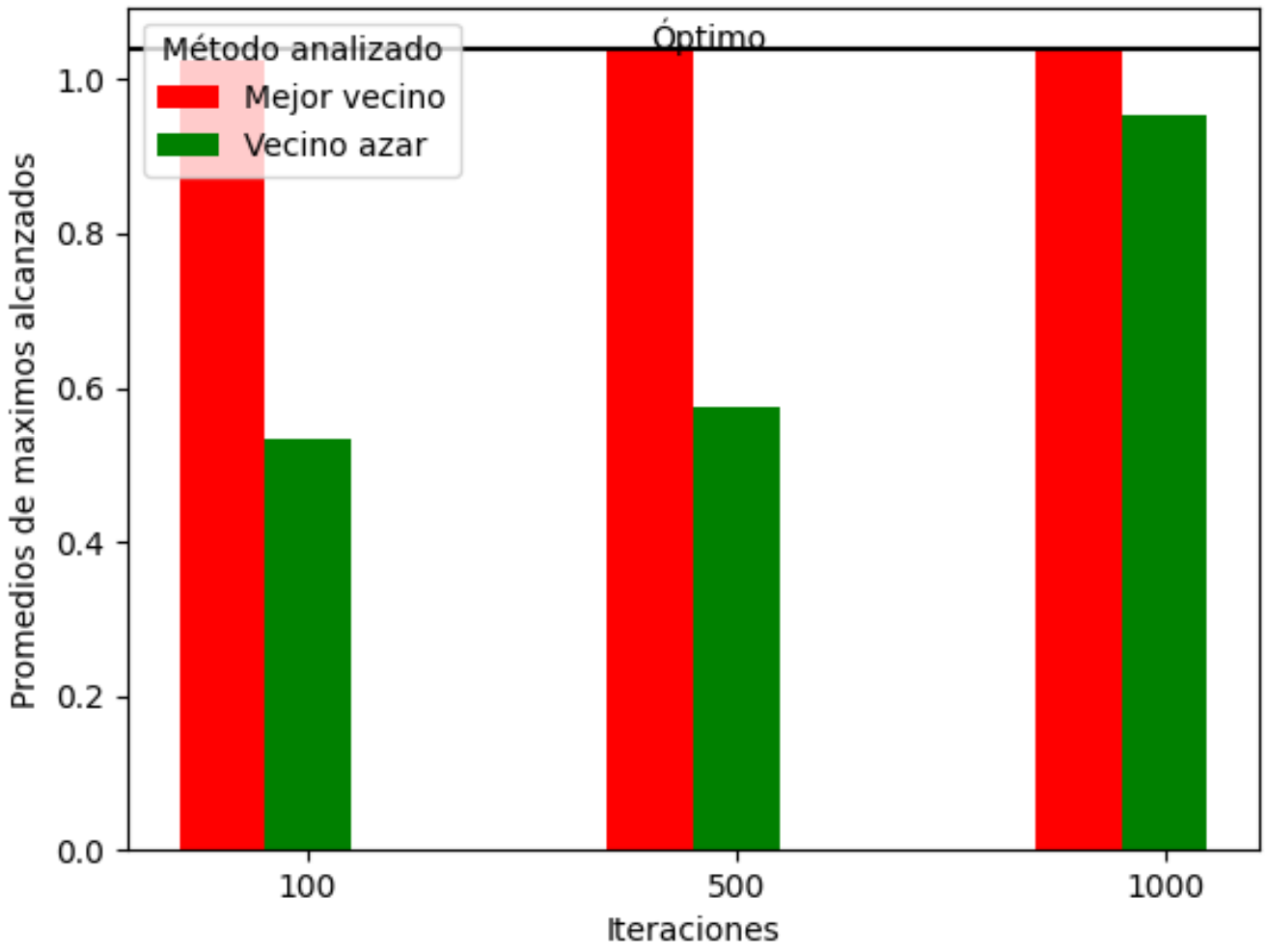


Figura 4: Diagrama de comparación de iteraciones

Referencias

[1] E. Schaeffer. Búsqueda local. *Repositorio, GitHub*, 2022. URL https://github.com/satuelisa/Simulation/blob/master/LocalSearch/minimize1D.py.

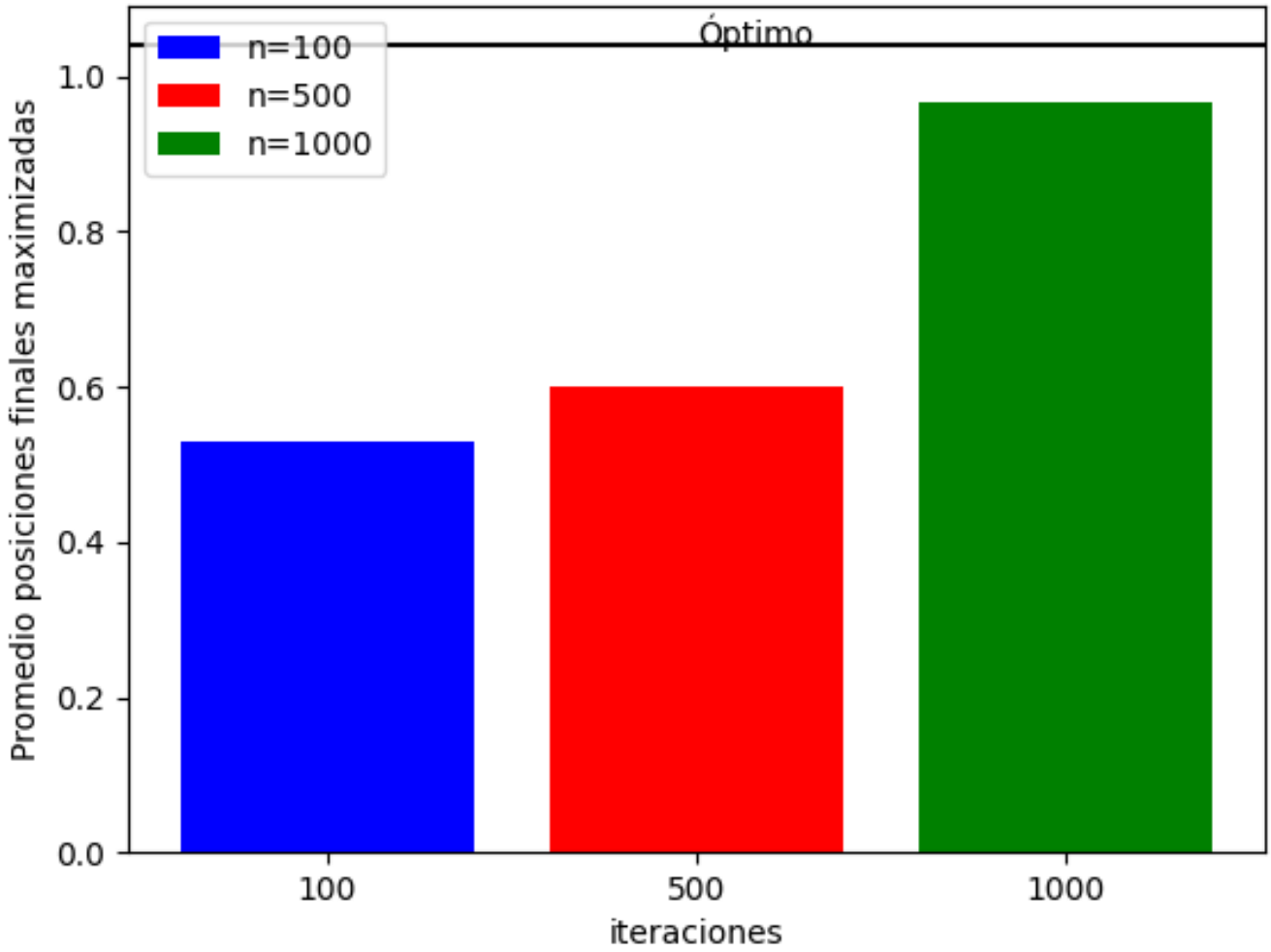


Figura 5: Diagrama promedio de posiciones de vecinos al azar

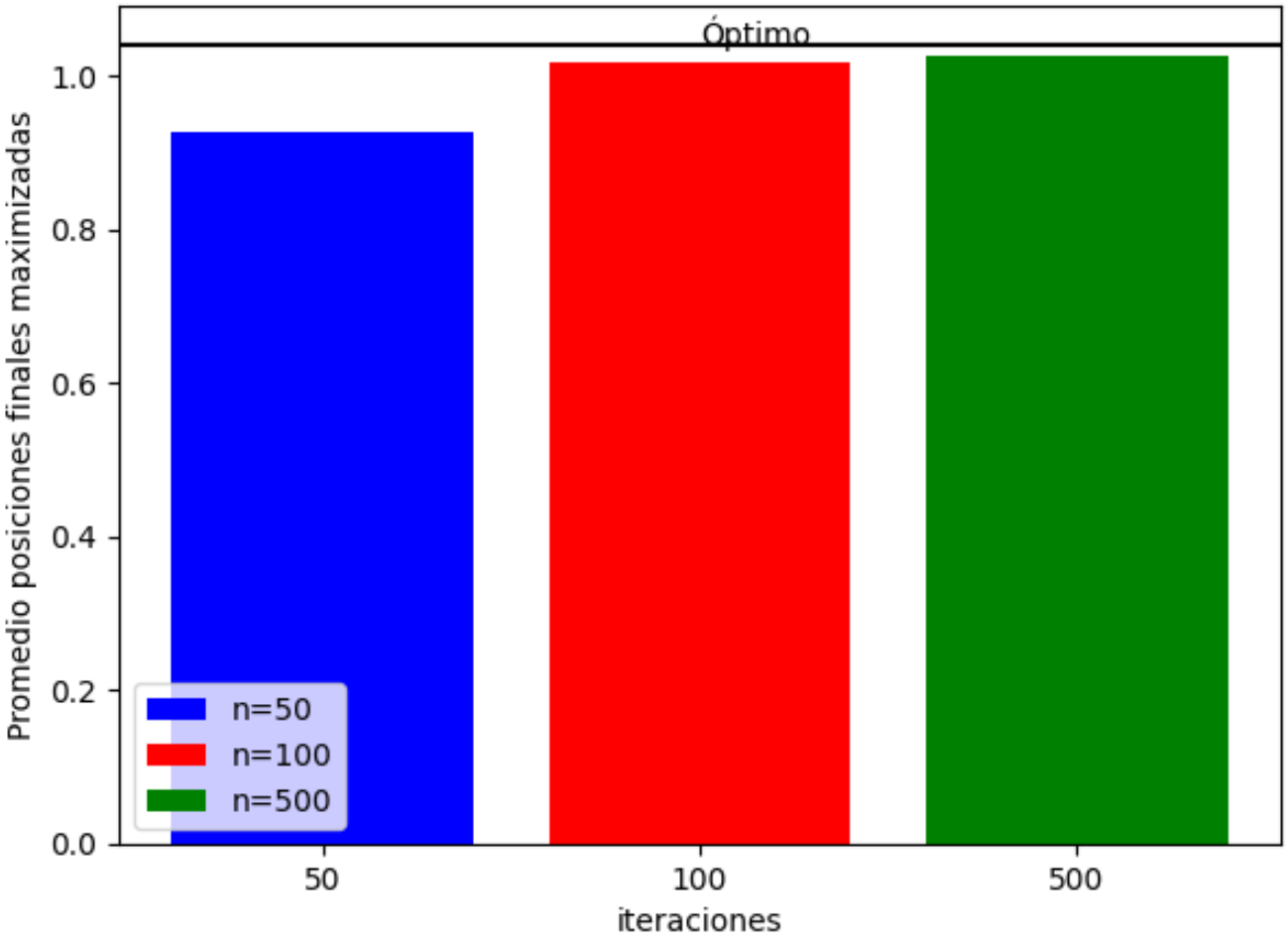


Figura 6: Diagrama promedio de posiciones de vecinos máximos

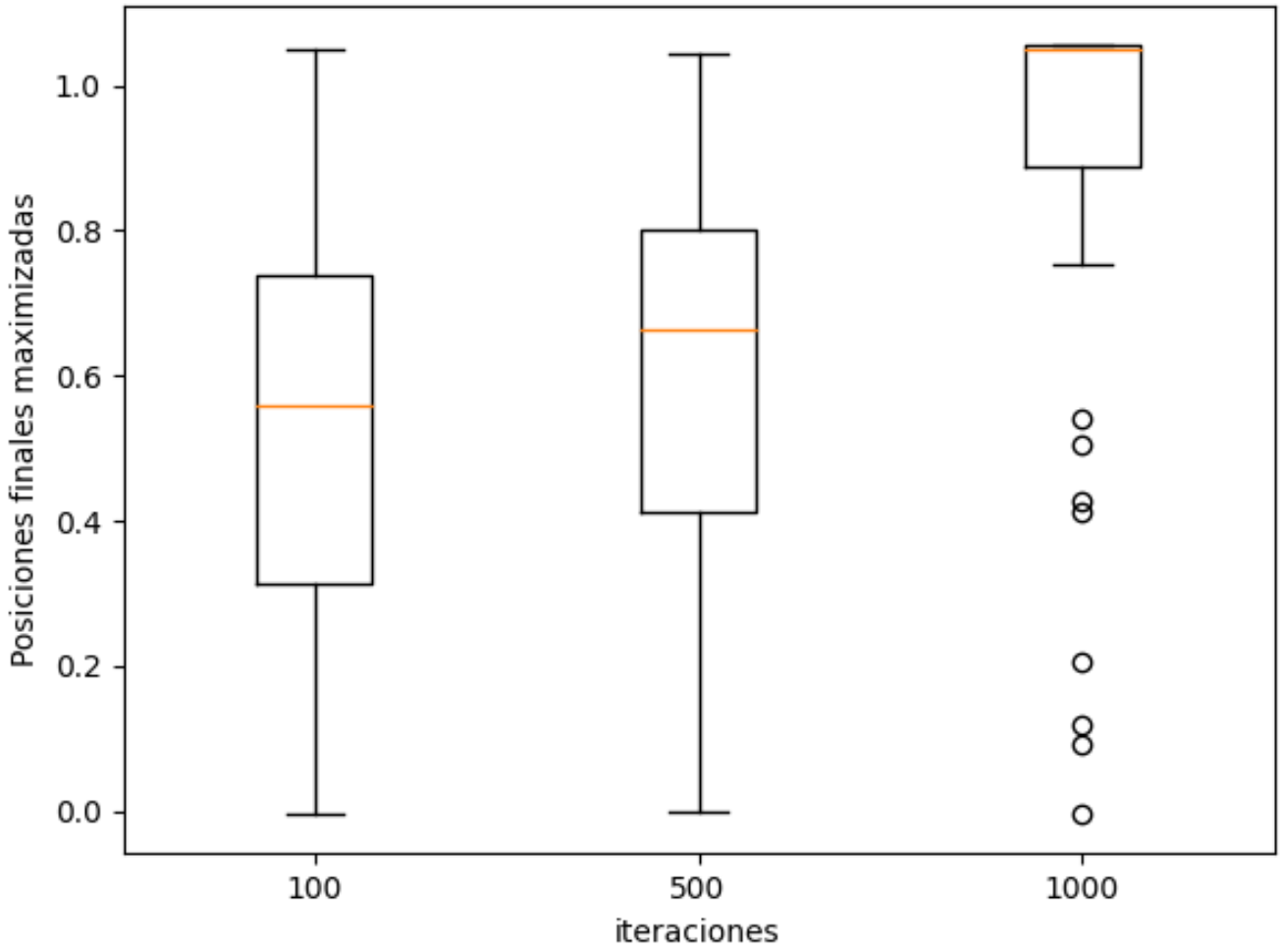


Figura 7: Diagrama de vecinos al azar

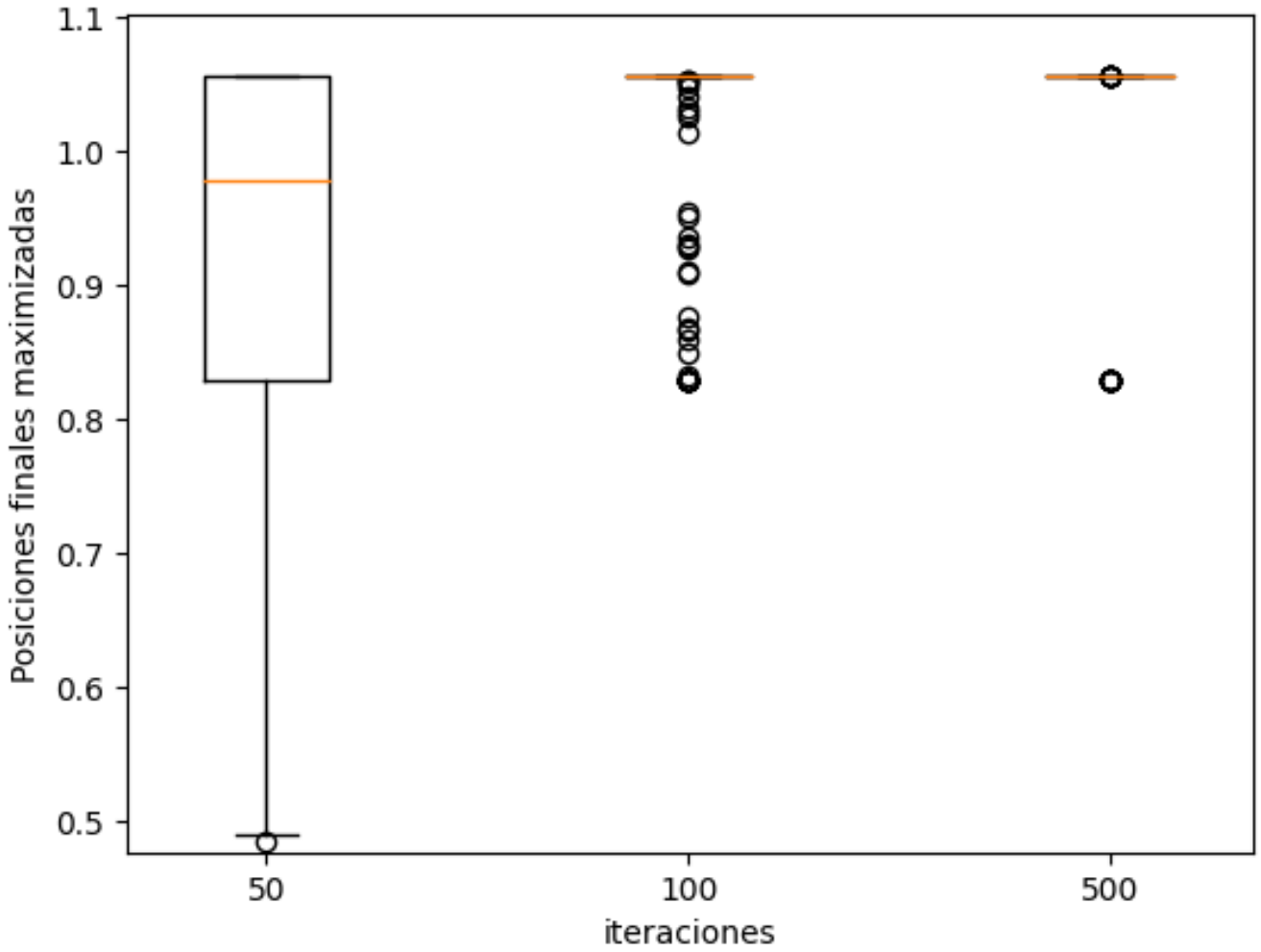


Figura 8: Diagrama de vecinos máximos