Problema 2

Optimización con Inteligencia de Enjambre

IWG-101

Departamento de Informática
Universidad Técnica Federico Santa María

2018-2

Contexto

Optimización

Un problema de optimización consiste en maximizar o minimizar una función escogiendo sistemáticamente valores de entrada, tomados de un conjunto definido, y calculando el valor de la función.

Metaheurística

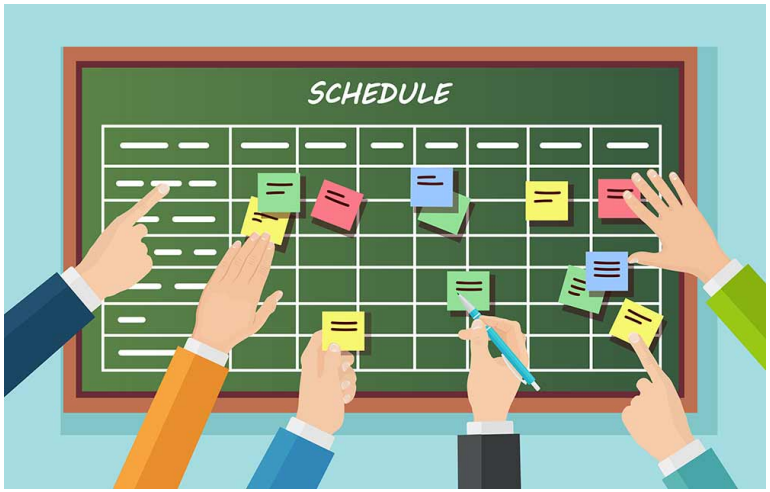
Método heurístico para resolver un tipo de problema computacional, utilizando parámetros dados por el usuario sobre unos procedimientos genéricos y abstractos de una manera (ojalá) eficiente.

Inteligencia de Enjambre

Rama de la **Inteligencia artificial** que estudia el comportamiento colectivo de los sistemas descentralizados, auto-organizados, naturales o artificiales.

Aplicaciones

Planificación:

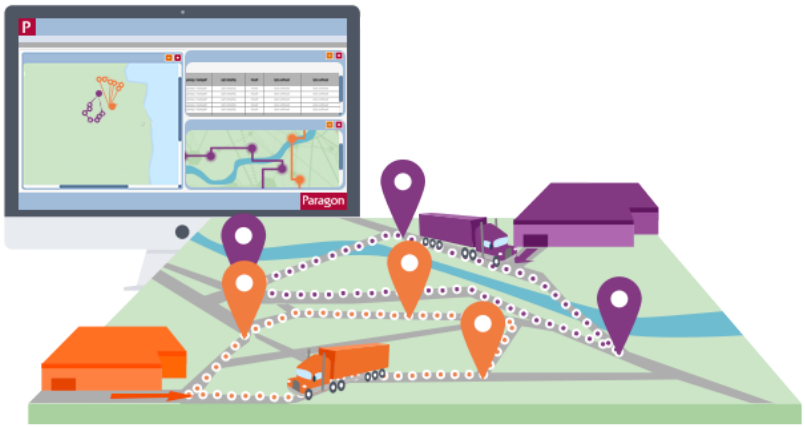


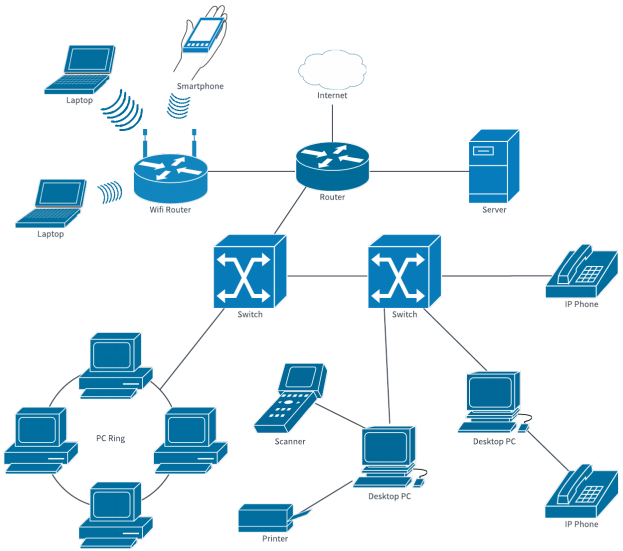
Aplicaciones

Optimización de procesos:

Aplicaciones

Enrutamiento:





Particle Swarm Optimización (PSO)

PSO [Kennedy, 1995]

Definición

Método computacional que optimiza un problema iterativamente mejorando la solución candidata respecto a una medida de calidad. Mediante una población de soluciones candidatas (partículas), se mueven en un espacio de búsqueda utilizando fórmulas simples de posición y velocidad. El movimiento de las partículas depende de la influencia de las partículas vecinas.

Esquema del PSO

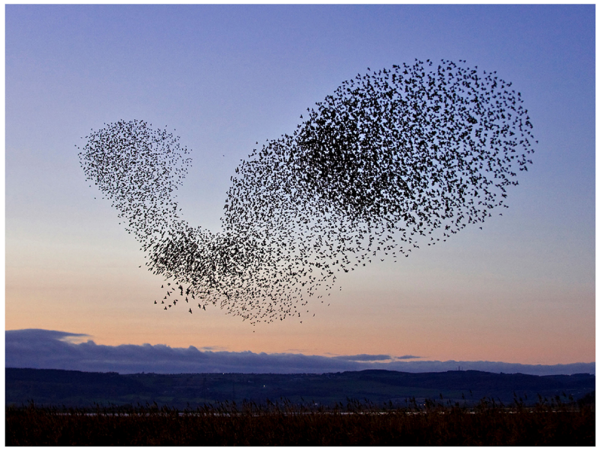


Figura: Bandada de pájaros[1].

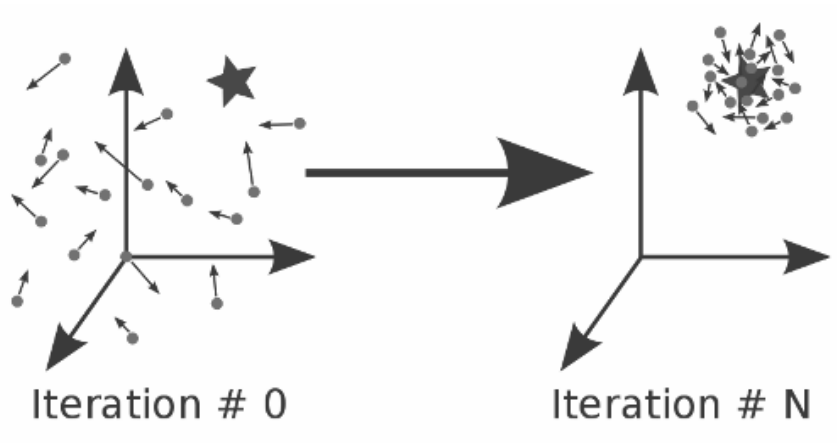


Figura: Esquema[2].

[1] *Fuente: https://esa.github.io/pagmo2/docs/cpp/algorithms/pso.html*

[2] *Fuente: ttps://www.frontiersin.org/articles/10.3389/fendo.2014.00239/full*

Algoritmo

Parámetros del algoritmo

Sea $f : \mathbb{R}^n \to \mathbb{R}$ la función de costo a ser optimizada. Sea N el número de partículas a utilizar, $(\mathbf{b}_{inf}, \mathbf{b}_{sup})$ los límites asociados al dominio del problema. Los parámetros ω, c_1, c_2 se asocian a la inercia, término cognitivo y social respectivamente.

Algorithm 1 Algoritmo PSO (parte 1).

Require: $f, N, \mathbf{b}_{inf}, \mathbf{b}_{sup}, \omega, c_1, c_2$

1: **for** $i = 1$ **to** N **do** /* Para cada partícula */
2: $\quad \mathbf{x}_i \sim \mathcal{U}(\mathbf{b}_{inf}, \mathbf{b}_{sup})$ /* Inicializar la posición de la partícula */
3: $\quad \mathbf{p}_i \leftarrow \mathbf{x}_i$ /* Inicializar la mejor posición de la partícula */
4: $\quad$ **if** $f(\mathbf{p}_i) < f(\mathbf{g})$ **then** /* Si la evaluación de la mejor posición es mejor que la global */
5: $\quad\quad \mathbf{g} \leftarrow \mathbf{p}_i$ /* Actualizar la mejor posición conocida por el enjambre */
6: $\quad$ **end if**
7: $\quad \mathbf{v}_i \sim \mathcal{U}(-|\mathbf{b}_{sup} - \mathbf{b}_{inf}|, |\mathbf{b}_{sup} - \mathbf{b}_{inf}|)$ /* Inicializar la velocidad de la partícula */
8: **end for**

Algoritmo

Algorithm 1 Algoritmo PSO (parte 2).

while No se cumpla el criterio de término **do**
  for $i = 1$ **to** N **do** /* Para cada partícula */
    for $d = 1$ **to** n **do** /* Por cada dimensión */
      $r_p, r_g \sim \mathcal{U}(0,1)$ /* Generar números aleatorios */
      $\mathbf{v}_{i+1}^d \leftarrow \omega\, \mathbf{v}_i^d + c_1\, r_p \left(\mathbf{p}_i^d - \mathbf{x}_i^d\right) + c_2\, r_g \left(\mathbf{g}^d - \mathbf{x}_i^d\right)$ /* Actualizar la velocidad de la partícula */
    end for
    $\mathbf{x}_{i+1} \leftarrow \mathbf{x}_i + \mathbf{v}_i$ /* Actualizar la posición de la partícula */
  end for
  if $f(\mathbf{x}_i) < f(\mathbf{p}_i)$ **then** /* Si la evaluación de la posición es mejor que la mejor posición */
    $\mathbf{p}_i \leftarrow \mathbf{x}_i$ /* Actualizar la mejor posición conocida para la partícula */
    if $f(\mathbf{p}_i) < f(\mathbf{g})$ **then** /* Si la evaluación de la mejor posición es mejor que la global */
      $\mathbf{g} \leftarrow \mathbf{p}_i$ /* Actualizar la mejor posición conocida por el enjambre */
    end if
  end if
end while
return g /* Retornar la mejor posición global obtenida */

Función Gaussiana

Función Gaussiana

La Gaussiana es una función de la forma:

$$f(x) = a \cdot \exp\left(-\frac{(x-b)^2}{2 \cdot c^2}\right), \tag{1}$$

para $a, b, c \in \mathbb{R}$ arbitrarios. Se denomina así en honor al notable matemático alemán Carl Friedrich Gauss. Esta función es reconocida por su forma de "campana", donde el parámetro a tiene relación con la altura, b es la posición del centro de la función y c controla el ancho de esta campana. Estas funciones son ampliamente utilizadas en estadística, procesamiento de señales e imágenes, entre otras.

Función Gaussiana

Para este problema en particular, utilizaremos una función Gaussiana en 2 dimensiones

$$f(x,y) = A \cdot \exp\left(-\left(\frac{(x-x_0)^2}{2\cdot\sigma_x^2} + \frac{(y-y_0)^2}{2\cdot\sigma_y^2}\right)\right), \tag{2}$$

donde $(x_0, y_0) \in \mathbb{R}^2$ corresponde al centro de la función, y $A, \sigma_x^2, \sigma_y^2 \in \mathbb{R}$ corresponden a los parámetros de altura y ancho para cada eje respectivamente.

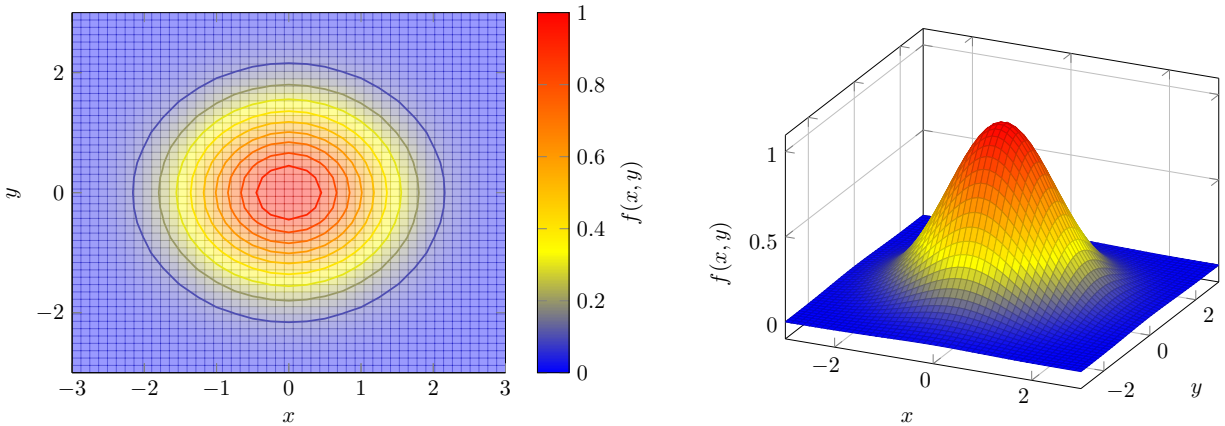


Figura: Función Gaussiana en 2D

Problema: Instalación de antenas

Instalación de antenas

Problema: Encontrar las mejores ubicaciones para instalar N antenas según distintos casos para diferentes escenarios.

Figura: Antena de celular

Ustedes tendrán 2 casos donde dispondrán de distintas fuentes de información:

1. Tendrán la información solo de las antenas ya existentes.
2. Tendrán la información tanto de las antenas ya existentes como de la ubicación de la población.

Caso 1: Solo información de las antenas.

Supondremos que la señal de las antenas es una combinación de funciones Gaussianas:

$$signal(x, y) = \sum_{i=1}^{N_{antennas}} \exp\left(-\left(\frac{(x - x_i)^2}{p_i} + \frac{(y - y_i)^2}{p_i}\right)\right) \quad (3)$$

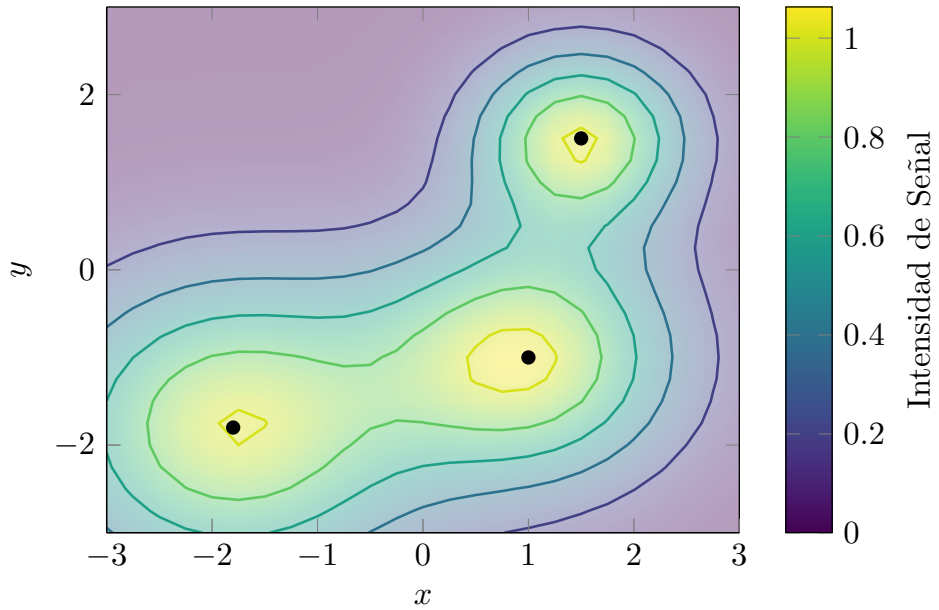


Figura: Ejemplo señal de antenas.

Caso 1: Solo información de las antenas.

- La información de las antenas la obtendrán de un archivo de texto llamado: `antenas_X.txt`, donde `X` será el identificador del escenario.
- El archivo de texto tendrá la información de todas las antenas existentes de la forma: `x,y,p`, donde `x, y` es la posición de la antena y `p` es la potencia.

La información extraída del archivo de texto debe ser almacenada en una lista de tuplas del tipo: `l_antenas = [(x,y,p), ...]`

Caso 2: Información tanto de antenas como de la población

Supondremos que la población esta distribuida como, $X \sim \mathcal{N}(\mu, \Sigma)$, donde μ es la media y Σ es la dispersión.

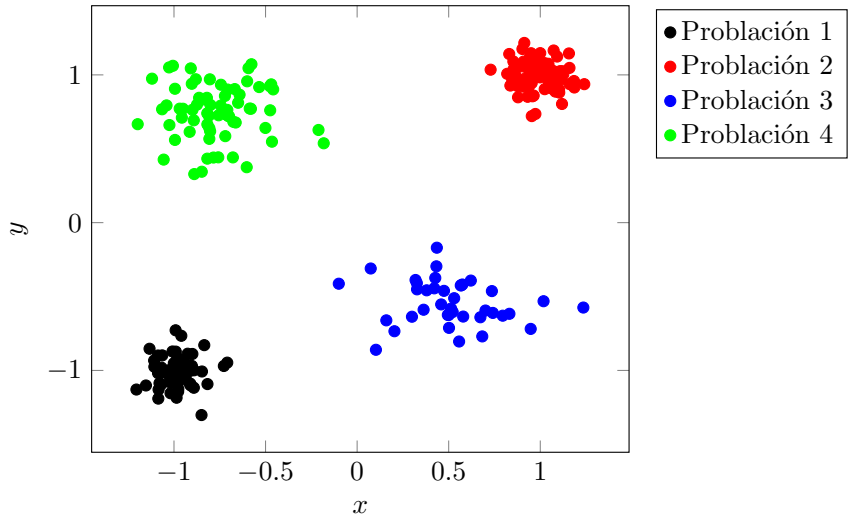


Figura: Ejemplo de población.

La función de distribución queda definida como:

$$population(x, y) = \sum_{i=1}^{N_{populations}} \exp\left(-\left(\frac{(x - c_{x,i})^2}{r_{1,i}} + \frac{(y - c_{y,i})^2}{r_{2,i}}\right)\right) \quad (4)$$

Caso 2: Información tanto de antenas como de la población

- La información de la población la obtendrán de un archivo llamado: `población_X.txt`, donde `X` será el identificador del escenario.
- El archivo de texto tendrá la información según la siguiente estructura:

```
nombre_localidad_1,c_x,c_y,r1,r2
x1, y1
x2, y2
...
nombre_localidad_2,c_x,c_y,r1,r2
x1, y1
...
```

donde se muestra el nombre de la localidad, la posición central de la localidad según la distribución de la población y los radios en los eje x e y respectivamente. Luego la posición de cada persona de esa localidad. El número de personas puede ser variable.

Caso 2: Información tanto de antenas como de la población

La información extraída del archivo de población debe ser almacenada en un diccionario de diccionarios del tipo:

```
d_poblacion = {nombre_localidad:{'pos':(c_x,c_y),
'radio': (r1, r2), 'poblacion':[(x1,y1), (x2, y2), ...]},
...
}
```

donde cada valor de la llave del diccionario es a su vez otro diccionario que guarda la información completa de cada localidad.

La llave `poblacion` tiene como valor una lista de tuplas con todas las ubicaciones de las personas de esa localidad.

Funciones a implementar

```
def leerAntenas(nombre_archivo):
    # Abrir archivo nombre_archivo y retornar
    # una lista de tuplas (antenas)
    return antenas

def leerPoblacion(nombre_archivo):
    # Abrir archivo nombre_archivo y retornar
    # un diccionario (poblacion)
    return poblacion

def buscarPorSenal(N, antenas):
    # Buscar N nuevas antenas, conociendo antenas
    # y retornar la nueva informacion optimizando
    # la funcion correspondiente.
    return nuevas_antenas
```

Funciones a implementar

```
def buscarPorPoblacion(N, poblacion, antenas):
    # Buscar N nuevas antenas, conociendo antenas
    # y retornar la nueva informacion optimizando
    # la funcion correspondiente.
    return nuevas_antenas

def graficarPoblacionAntenas(poblacion, antenas):
    # Graficar informacion de poblacion y antenas.
    # Puede guiarse utilizando 'utilities'
    # No retorna nada
```

Funciones proporcionadas

Revisar la documentación en `utilities.py`.

Escenarios

- Existen 2 escenarios, los cuales tendrán sus respectivos archivos: `antenas_1.txt`, `poblacion_1.txt` y `antenas_2.txt`, `poblacion_2.txt`.
- En ambos escenario el límite del dominio es $[0, 10] \times [0, 10]$.
- El primer escenario es más sencillo que el segundo y deberán resolver ambos casos respondiendo las preguntas entregadas a continuación.

Preguntas

Deben responder las siguiente preguntas para cada escenario:

1. Para cada caso:
   1. ¿Cómo varia el comportamiento del algoritmo PSO según el número de partículas?.
   2. ¿Como varia el comportamiento del algoritmo PSO al modificar el valor de los parámetros que calculan la velocidad de las partículas?
2. ¿Cuál fuente de información es más importante?
3. Si tuvieran que pagar por la información de la población: ¿cuánto pagarían?

Preguntas++

1. Si ahora no tienen un número de antenas fijas que ubicar:
    1. ¿Qué harían ustedes para resolver el problema?, si les piden minimizar los costos, pero sin perjudicar la calidad de servicio.
    2. ¿Cuáles supuestos creen que son más determinantes al momento de resolver el problema?
    3. Intente resolver la variante expuesta!!

En resumen...

1. Instalar las librerías necesarias para solucionar el problema y ejecutar el código facilitado:
   - `NumPy`: `http://www.numpy.org`
   - `Matplotlib`: `https://matplotlib.org`
2. Analizar el código entregado y pensar como resolver el problema.
3. Implementar las funciones solicitadas: `leerAntenas`, `leerPoblacion`, `buscarPorSenal`, `buscarPorPoblacion`, `graficarPoblacionAntenas`.
4. Preparar una presentación en Jupyter Notebook que incluya todo el código fuente y las conclusiones del trabajo (la presentación debe permitir modificar código y parámetros en vivo).
5. Exponer su solución en clases.

Condiciones de Entrega

Entrega en `http://moodle.inf.utfsm.cl`

El trabajo completo debe ser desarrollado en **Jupyter Notebook**, incluyendo el código y la presentación.

Fecha de Entrega

Una semana a partir de la presentación del problema, es decir, la semana del **01/10/2018**. Deberán subir a moodle su desarrollo antes de que el primer grupo realice su presentación.

Presentación

Todos los integrantes del equipo deben presentar su solución en clase.

References

J. Kennedy and R. Eberhart (1995)
Particle swarm optimization
Proceedings of ICNN'95 - International Conference on Neural Networks 4, 1942 – 1948.

Otra referencia:
Wikipedia