Algoritmos Genéticos Adaptativos para la Optimización de la Distancia Mínima entre Paneles Solares

John Sebastián Galindo Hernández, Miguel Ángel Moreno Beltrán

***Abstract*—This paper presents an adaptive genetic algorithm (AGA) to optimize the minimum spacing between A-135P photovoltaic panels (1476×659×35mm) installed at latitude 41°with a fixed tilt of 45°. Two seasonal scenarios are considered—winter (critical case) and summer (permitting reduced spacing). Dynamic adjustment of crossover and mutation rates based on population diversity or fitness improvement is implemented to enhance convergence.**

***Resumen*—Este artículo presenta un algoritmo genético adaptativo (AGA) para optimizar la distancia mínima entre paneles fotovoltaicos A-135P (1476×659×35 mm) instalados a latitud 41°con inclinación fija de 45°. Se consideran dos escenarios estacionales—invierno (caso crítico) y verano (con posible reducción de distancia). Se implementa ajuste dinámico de probabilidades de cruce y mutación basado en diversidad de población o mejora de fitness para mejorar la convergencia.**

***Index Terms*—algoritmos genéticos adaptativos, optimización, energía solar, distancia entre paneles, computación evolutiva**

I. Introducción

El siguiente informe aborda la optimización del diseño de un panel solar producido por la planta solar Guayepo en el que se busca maximizar su eficiencia energética bajo condiciones específicas como lo son el angulo de inclinación, la orientación, el tipo de material, y otros factores tanto físicos como ambientales. La metodología del presente informe se fundamenta en la adaptación del código de un algoritmo genético realizado en la pasada clase de Machine Learning y algoritmos genéticos. Este informe se centra específicamente en dos variables de las tantas que afectan el diseño del panel solar, las cuales son el ángulo de inclinación y la orientación, estas variables se ven representadas como un cromosoma de 10 bits donde se dividirá en 5 bits para cada variable, por otra parte, se escoge una función con forma paraboloide con un único óptimo global como función objetivo.

II. Fundamentos Teóricos

A. Declinación Solar

El concepto de declinación solar, δ, , se refiere al ángulo formado entre los rayos del Sol y el plano horizontal correspondiente al ecuador terrestre. A causa de esta inclinación en el eje de la Tierra (23.45°), este ángulo cambia estacionalmente en $\pm\,23{,}45°$ alcanzando los valores más grandes (23.45°) en los solsticios de verano e invierno [1]. El ángulo con valor de $\pm 23{,}45°$ esta relacionado con la oblicuidad de la eclíptica (la línea curva por la que parece .ªndar.el Sol alrededor de la Tierra).

B. Elevación Solar

Este ángulo se refiere a la elevación solar más alta que se puede obtener al mediodía, α, depende de la latitud φ y de la declinación δ. En un día dado:

$$\theta_s = |\varphi - \delta| \quad \Longrightarrow \quad \alpha = 90° - \theta_s = 90° - |\varphi - \delta|.$$

Para $\delta \leq \varphi$ (norte), esto se simplifica a

$$\alpha = 90° - \varphi + \delta,$$

que es la forma usada para este ejercicio a modo de simplificación garantizando que la altura del Sol se calcule correctamente en función de la estación del año [2].

C. Distancia Mínima entre Paneles

Para evitar sombras, la distancia mínima D_M entre filas de paneles de longitud B y ángulo de inclinación β se obtiene de la proyección horizontal del panel más el espacio requerido para que su sombra no alcance al siguiente:

$$D_M = B\cos\beta + \frac{B\sin\beta}{\tan\alpha}.$$

El primer término, $B\cos\beta$, es la proyección horizontal de la longitud del panel; el segundo, $B\sin\beta/\tan\alpha$, proviene de la relación geométrica entre la altura del borde superior del panel ($B\sin\beta$) y el ángulo crítico de elevación α [3].

D. Función de Aptitud (Fitness)

En este ejercicio práctico se define la función de aptitud como la diferencia absoluta con respecto al valor teórico ideal D_M:

$$\text{aptitud}(D, D_M) = |D - D_M|.$$

Se utiliza a menudo este pauta (error absoluto) en los algoritmos genéticos porque penaliza las sobreestimaciones y subestimaciones, y premia convergencia directa hacia el valor óptimo necesario [4].

E. Operadores de Cruce

Para poder realizar más pruebas sobre el funcionamiento del algoritmo genético, se decidió usar varios métodos de cruce, siendo así el método seleccionable de manera dinámica por el ejecutor del código.

a) Cruce Aritmético: (Arithmetic Crossover) Este método permite generar los hijos dentro del segmento que une a los padres:

$$H_1 = \alpha P_1 + (1-\alpha)P_2, \quad H_2 = (1-\alpha)P_1 + \alpha P_2, \quad \alpha \in [0,1].$$

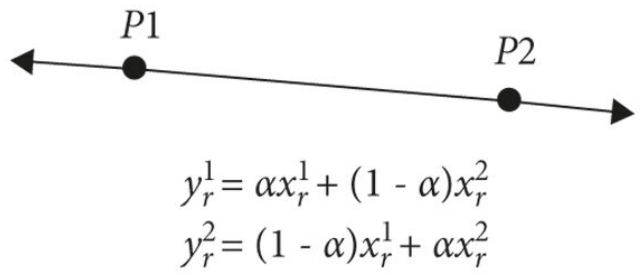

Figura 1. Representación del rango en el cruce aritmético [5]

Es sencillo y explota la información de ambos padres, aunque por otro lado se puede reducir la diversidad si el valor de α es siempre constante [4].

b) BLX-α (Blend Crossover): Este método extiende el intervalo $[P_{\text{mín}}, P_{\text{máx}}]$ en un factor α y muestrea uniformemente:

$$H_i \sim U\big(P_{\text{mín}} - \alpha\Delta,\ P_{\text{máx}} + \alpha\Delta\big), \quad \Delta = P_{\text{máx}} - P_{\text{mín}}.$$

Esto permite explorar fuera del rango padre, controlado por mediante la pertenencia de $\alpha \in (0,1)$. El unico problema es cuando α tiende a un valor muy grande por que los hijos se alejan mucho agregando ruido.

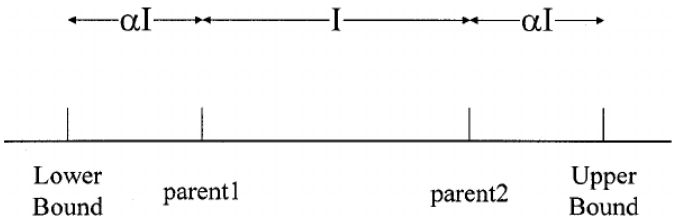

Figura 2. Intervalo de un cruce BLX- α. [6]

c) 3. SBX (Simulated Binary Crossover): Por último, este tercer método imita cruces binarios en reales usando un índice de distribución η:

Primero, se necesita que los hijos 'caigan' alrededor de los padres, lo cual se logra con una probabilidad análoga de distribución, esto se realiza con el término η en la densidad de probabilidad para β

$$\beta = \begin{cases} (2u)^{1/(\eta+1)}, & u \le 0{,}5, \\ \big[2(1-u)\big]^{-1/(\eta+1)}, & u > 0{,}5, \end{cases}$$

Luego, se haya el valor para cada hijo:

$$H_1 = 0{,}5\big[(1+\beta)P_1 + (1-\beta)P_2\big]$$
$$H_2 = 0{,}5\big[(1-\beta)P_1 + (1+\beta)P_2\big].$$

Se balancea la exploración y explotación según $\eta \in [2,5]$ normalmente.

F. Operadores de Mutación

a) 1. Mutación Gaussiana: Se añade ruido normal truncado para no exceder límites:

$$D' = \text{mín}\big(\text{máx}(D + \mathcal{N}(0,\sigma), D_{\text{mín}}), D_{\text{máx}}\big).$$

Control fino de la magnitud de la perturbación con σ [7].

b) 2. Mutación Polinómica: Introduce variaciones pequeñas según un índice η_m (Deb & Agrawal):

$$\delta = \begin{cases} (2u)^{1/(1+\eta_m)} - 1, & u < 0{,}5, \\ 1 - \big[2(1-u)\big]^{1/(1+\eta_m)}, & u \ge 0{,}5, \end{cases}$$

$$D' = \begin{cases} D + \delta\,(D - D_{\text{mín}}), & \delta < 0, \\ D + \delta\,(D_{\text{máx}} - D), & \delta \ge 0. \end{cases}$$

Con $\eta_m \approx$ 20–100 se obtiene buen compromiso exploración/explotación [7].

G. Adaptación de Parámetros

En los AGAs se ajustan dinámicamente p_m y p_c para mantener diversidad y velocidad de convergencia [8] :

a) 1. Basado en Diversidad: La idea es medir la diversidad D como la desviación estándar media de cada individuo en la población, por lo que las probabilidades de mutación (p_m) y cruce (p_c) se ajustan según:

$$p_m(D) = p_{m,\text{mín}} + (p_{m,\text{máx}} - p_{m,\text{mín}})\, e^{-k_m D}$$
$$p_c(D) = p_{c,\text{máx}} - (p_{c,\text{máx}} - p_{c,\text{mín}})\, e^{-k_c D}.$$

Entonces cuando la población es muy homogénea (D es pequeño), P_m tiende hacia su valor máximo y P_c tiende a menos valor para menor cruce, por el contrario si la población es diversa el valor de la probabilidad de cruce aumenta y la de mutación disminuye.

b) 2. Basado en Mejora de Fitness: En este método la idea es calcular la mejora relativa del fitness promedio entre generaciones:

$$\Delta F = (f_{t-1} - f_t)/f_{t-1}$$

luego se ajusta con:

$$p_m(\Delta F) = p_{m,\text{mín}} + (p_{m,\text{máx}} - p_{m,\text{mín}})(1 - \Delta F),$$
$$p_c(\Delta F) = p_{c,\text{mín}} + (p_{c,\text{máx}} - p_{c,\text{mín}})\Delta F.$$

con esto se logra que si la mejora es pequeña ($\Delta F \to 0$) sube la probabilidad de mutación y baja la probabilidad de cruce, si la mejora es grande ($\Delta \to F1$) entonces se centra mas en la explotación subiendo p_c y bajando p_m [9].

III. Metodología

Teniendo en cuenta que el objetivo del ejercicio práctico es calcular la distancia mínima entre paneles solares para evitar sombras y maximizar la captación de energía en condiciones estacionales (verano o invierno), se adapta un algoritmo genético (AG) básico aprendido en clase. Se mantiene la estructura original con los bloques esenciales: generación de población, función *fitness*, selección, cruce, mutación y reemplazo. Como experimento adicional, se incorporan múltiples variantes de operadores de cruce, mutación y estrategias de adaptación de parámetros, para evaluar su impacto en la convergencia y robustez del AG.

A. Representación y población inicial

- **Individuo**: un número real $D \in [0, 10000]$ (distancia en mm).
- **Población inicial**: se genera con

```
def create_individual(valor_min, valor_max
    ):
    return random.uniform(valor_min,
        valor_max)

def create_poblation(len_poblation,
    valor_min, valor_max):
    return [create_individual(valor_min,
        valor_max)
            for _ in range(len_poblation)]
```

donde `len_poblation` es el tamaño de población (por defecto 10).

B. Cálculo de la distancia objetivo

La distancia teórica D_M se calcula a partir de la latitud, la inclinación fija del panel con un angulo de $\beta = 45°$ y la declinación estacional solar δ (±23.45°):

```
def min_angle(latitude, solar_decline):
    return 90 - latitude + solar_decline

def min_distance(panel_dimensions, latitude,
                 inclination_degree,
                     solar_decline):
    B = panel_dimensions[0]
    beta = math.radians(inclination_degree)
    alpha_min = math.radians(min_angle(
        latitude,
                                        solar_decline
                                            ))
    return B * math.cos(beta) + \
           (B * math.sin(beta)) / math.tan(
               alpha_min)
```

Este valor sirve como *target* para la función de aptitud.

C. Función de aptitud

Se define como el error absoluto respecto al valor teórico a modo de hallar la menor diferencia entre los individuos y la distancia ideal:

$$\text{fitness}(D, D_M) = \big| D - D_M \big|.$$

Menor *fitness* indica individuo más cercano a la solución óptima:

```
def fitness(x, target_distance):
    return abs(x - target_distance)
```

D. Selección

Se emplea selección por torneo simple formando un subgrupo con un % de los individuos en la población:

```
def ranking_selection(poblation,
    target_distance,
                      sub_poblation_percentage
                          ):
    len_sub = int(len(poblation) *
        sub_poblation_percentage)
    idxs = random.sample(range(len(poblation))
        , len_sub)
    sub = [poblation[i] for i in idxs]
    return min(sub, key=lambda x:
                fitness(x, target_distance))
```

Aquí, `sub_poblation_percentage` controla el tamaño del sub-torneo (por defecto 0.5).

E. Operadores de cruce

Se implementan las tres variantes vistas en la fundamentación teórica:

Método por cruce aritmetico

```
def arithmetic_crossover(p1, p2, alpha=None):
    if alpha is None:
        alpha = random.random()
    return (alpha*p1 + (1-alpha)*p2,
            (1-alpha)*p1 + alpha*p2)
```

Método BLX - α

```
def blx_alpha(p1, p2, alpha=0.3):
    lo, hi = min(p1, p2), max(p1, p2)
    d = hi - lo
    return (random.uniform(lo - alpha*d,
                           hi + alpha*d),
            random.uniform(lo - alpha*d,
                           hi + alpha*d))
```

método de simulación de cruce binario

```
def sbx(p1, p2, eta=2):
    u = random.random()
    if u <= 0.5:
        beta = (2*u)**(1/(eta+1))
    else:
        beta = (1/(2*(1-u)))**(1/(eta+1))
    return (0.5*((1+beta)*p1 + (1-beta)*p2),
            0.5*((1-beta)*p1 + (1+beta)*p2))
```

F. Operadores de mutación

Para los métodos de mutación se implementaron las dos estrategias vistas en la fundamentación teórica:

Mutación Gaussiana:

```
def gaussian_mutation(D, D_min, D_max, sigma
    =500):
    Dp = D + random.gauss(0, sigma)
    return min(max(Dp, D_min), D_max)
```

Mutación Polinomica

```
def polynomial_mutation(D, D_min, D_max, eta_m
    =20):
    u = random.random()
    if u < 0.5:
        delta = (2*u)**(1/(1+eta_m)) - 1
    else:
        delta = 1 - (2*(1-u))**(1/(1+eta_m))
    if delta < 0:
        Dp = D + delta*(D - D_min)
    else:
        Dp = D + delta*(D_max - D)
    return min(max(Dp, D_min), D_max)
```

G. Adaptación de probabilidades

Para generar un AGA, las probabilidades de cruce p_c y mutación p_m se ajustan cada generación mediante dos métodos:

Adaptación según diversidad de población

```
def diversity_adaptation(poblation, pm_min,
    pm_max,
                         pc_min, pc_max, k_m,
                             k_c):
    D = statistics.pstdev(poblation)
    pm = pm_min + (pm_max - pm_min)*math.exp(-
        k_m * D)
    pc = pc_max - (pc_max - pc_min)*math.exp(-
        k_c * D)
    return pm, pc
```

Adaptación según tasa de mejora

```
# Fitness-based adaptation
fit_prev = avg_fitness(poblation, target)
fit_curr = avg_fitness(new_poblation, target)
deltaF = (fit_prev - fit_curr) / fit_prev
pm = pm_min + (pm_max - pm_min)*(1 - deltaF)
pc = pc_min + (pc_max - pc_min)*deltaF
```

H. Algoritmo completo

Finalmente, el bucle principal del AG se mantiene en el documento la versión que muestra lo esencial del funcionamiento ya que para la aplicación esta función se extiende para manejar el envío y recibimiento de parámetros a modo de personalización y visualización de la información.

```
def genetic_algorithm(...):
    sd = (summer_solar_decline
          if season.lower() in ["verano","
              summer"]
          else winter_solar_decline)
    target = min_distance(...)

    poblacion = create_poblation(...)
    fit_prev = (avg_fitness(poblacion, target)
                if diversity_method=="fitness"
                    else None)

    history, pop_history, best_history = [],
        [], []
    for gen in range(max_generations):
        pop_history.append(poblacion.copy())
        best_curr = ranking_selection(
            poblacion,
                                      target,
                                         1.0)
        best_history.append(best_curr)

        if diversity_method=="diversity":
            pm, pc = diversity_adaptation(...)
        else:
            # fitness-based adaptation
            ...

        nueva = []
        while len(nueva) < len_poblation:
            p1 = ranking_selection(poblacion,
                                   target,
                                      0.5)
            p2 = ranking_selection(poblacion,
                                   target,
                                      0.5)
            if random.random() < pc:
                h1, h2 = globals()[
                    crossover_method](
                          p1, p2,
                              crossover_param)
            else:
                h1, h2 = p1, p2

            if random.random() < pm:
                h1 = globals()[mutation_method
                    ](
                      h1, valor_min, valor_max,
                      mutation_param)
            if random.random() < pm:
                h2 = globals()[mutation_method
                    ](
                      h2, valor_min, valor_max,
                      mutation_param)

            nueva.extend([h1, h2])

        poblacion = nueva[:len_poblation]

    best = best_history[-1]
    return {
        "target": target,
        "best": best,
        "fitness": fitness(best, target),
        "history": history,
        "populations": pop_history,
        "bests": best_history
    }
```

I. Visualización de Resultados

Finalmente, en la creación del código para la actividad propuesta, se usa una interfaz web que permita a cualquier usuario acceder a esta funcionalidad y probar el resultado de el trabajo realizado para solucionar la problemática propuesta sobre la distancia entre paneles solares según su angulo y temporada. Esta parte Frontend se construyó con el Framework Streamlit con el fin de generar una visualización fácil, atractiva y simple de entender, aquí se proporcionó tanto un formulario básico como uno avanzado para la prueba de diferentes parámetros en el procesamiento que hace el algoritmo genético como también para los resultados se proporcionaron representaciones visuales de la distancia entre paneles y gráficos sobre la evolución del mejor individuo a lo largo de las generaciones y las variables de probabilidad de mutación y probabilidad de cruce a lo largo de las generaciones, por otro lado, como complemento se dan datos relevantes como la distancia mínima ideal y el valor de cada individuo en la población resultante en cada generación.

A continuación se muestran parte del Frontend final:

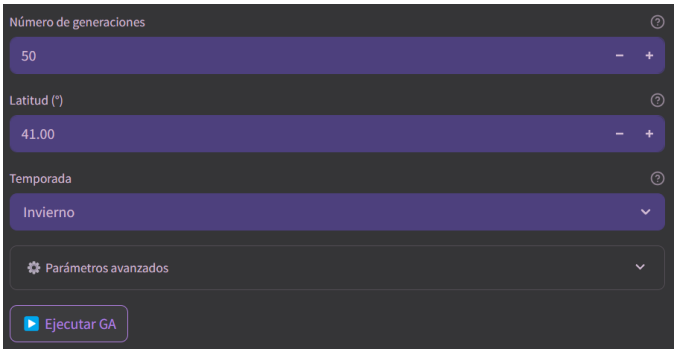

Figura 3. Parámetros Básicos

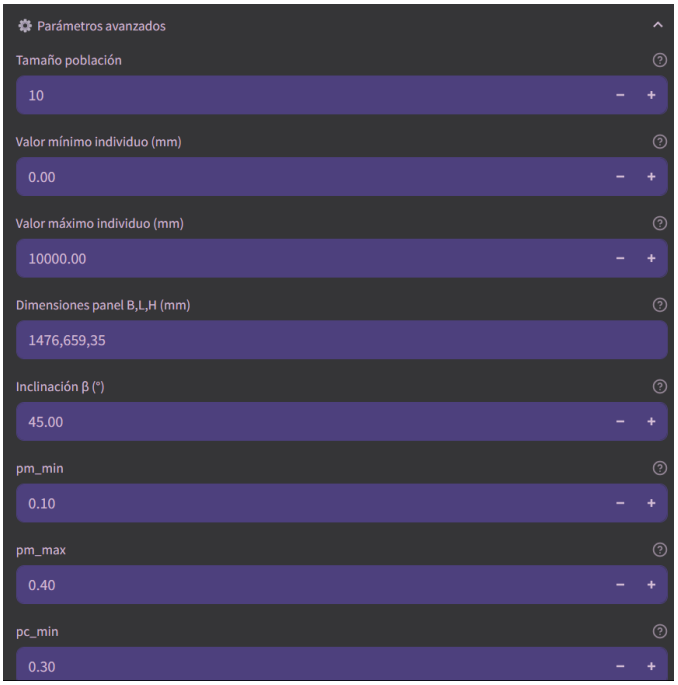

Figura 4. Parámetros Básicos

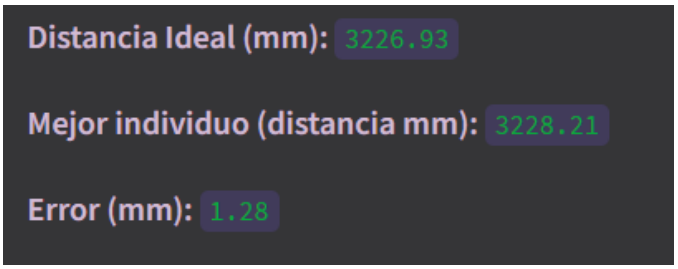

Figura 5. Resultados Clave

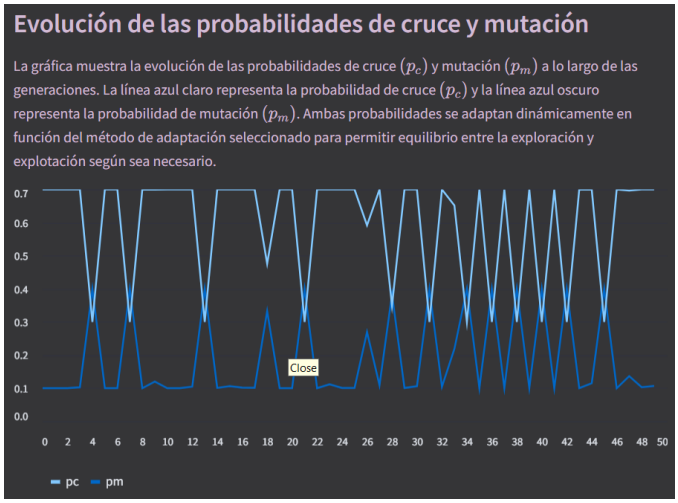

Figura 6. Gráfica de evolución de probabilidades (mutación y cruce)

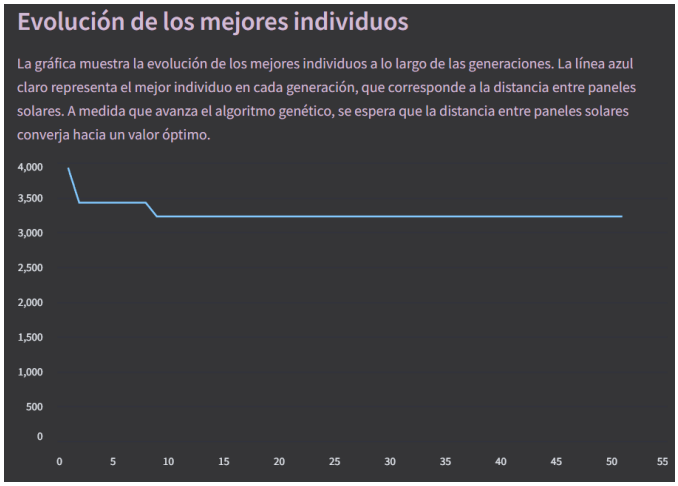

Figura 7. Gráfico de evolución del mejor individuo a través de las generaciones

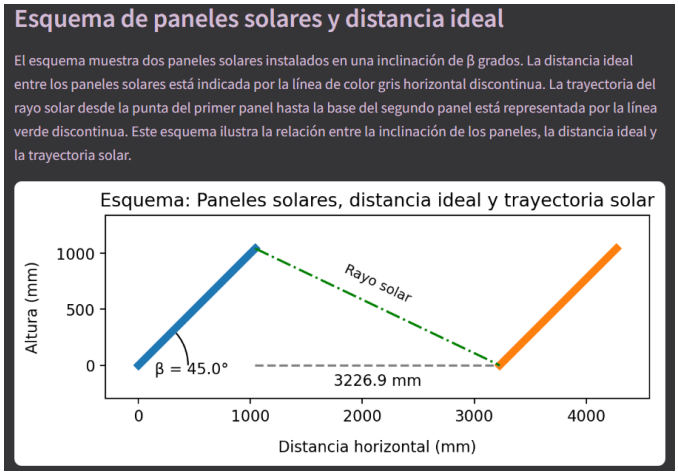

Figura 8. Representación de los paneles solares

IV. Resultados

Caso #1: Caso Base

- **Generaciones:** 100
- **Latitud:** 41°
- **Temporada:** Invierno
- **Objetivo:** Punto de referencia sobre el rendimiento del algoritmo.

Figura 9. Error durante generaciones Caso #1

Resultados Detallados de las Generaciones

Generación 1:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:6305.8
1:8000.63
2:4946.41
3:9382.32
4:6560.52
5:6672.94
6:5994.11
7:1776.21
8:2395.93
9:5597.03
]
```

Mejor individuo: 2395.93

Generación 99:

- **pc =** 0.1001
- **pc =** 0.7
- **Población:**

```
[
0:3226.47
1:3772.96
2:3226.47
3:3226.47
4:3226.47
5:3226.47
6:3226.47
7:3226.47
8:3226.47
9:3780.16
]
```

Mejor individuo: 3226.47

Generación 100:

- **pc =** 0.4
- **pc =** 0.3
- **Población:**

```
[
0:3226.47
1:3226.47
2:3226.47
3:3226.47
4:3226.47
5:3226.47
6:3226.47
7:3226.47
8:3226.47
9:3226.47
]
```

Mejor individuo: 3226.47

Caso #2: Optimización en Condiciones Favorables

- **Generaciones:** 200
- **Latitud:** 20°
- **Temporada:** Verano
- **Objetivo:** Prueba con la mayor cantidad de iteraciones, un ambiente con alta incidencia solar y en la mejor temporada. Esto debería demostrar el potencial máximo del algoritmo.

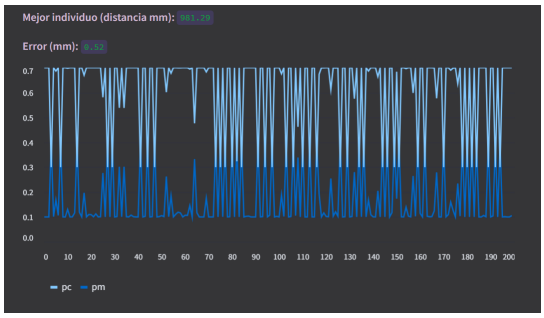

Figura 10. Error durante generaciones Caso #2

Generación 1:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:9997.36
1:668.66
2:9653.57
3:6704.05
4:4022.1
5:4402.86
6:4663.32
7:9158.88
8:9065.63
9:8334.01
]
```

Mejor individuo: 668.66

Generación 2:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:4402.86
1:4402.86
2:1674.69
3:3016.06
4:4288.63
5:4136.32
6:668.66
7:1173.65
8:4022.1
9:4022.1
]
```

Mejor individuo: 1173.65

Generación 199:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:981.29
1:981.29
2:981.29
3:981.29
4:981.29
5:0
6:981.29
7:981.29
8:981.29
9:981.29
]
```

Mejor individuo: 981.29

Generación 200:

- **pc =** 0.1052
- **pc =** 0.7
- **Población:**

```
[
0:594.44
1:981.29
2:981.29
3:981.29
4:981.29
5:981.29
6:981.29
7:981.29
8:981.29
9:981.29
]
```

Mejor individuo: 981.29

Caso #3: Prueba de Estrés en Condiciones Adversas

- **Generaciones:** 50
- **Latitud:** 60°
- **Temporada:** Invierno
- **Objetivo:** Situación crítica donde la baja incidencia y el alto ángulo hacen la optimización más difícil.

Figura 11. Error durante generaciones Caso #3

Generación 1:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:5428.14
1:4349.43
2:337.66
3:3832.89
4:7571.96
5:5315.55
6:9803.78
7:5887.19
8:6521.06
9:4969.63
]
```

Mejor individuo: 9803.78

Generación 2:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:9134.23
1:8241.51
2:7571.96
3:7571.96
4:8241.51
5:9134.23
6:8241.51
7:9134.23
8:7571.96
9:6521.06
]
```

Mejor individuo: 9134.23

Generación 49:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:10000
1:10000
2:9201.06
3:10000
4:9505.25
5:10000
6:10000
7:10000
8:10000
9:10000
]
```

Mejor individuo: 10000.0

Generación 50:

- **pc =** 0.1004
- **pc =** 0.7
- **Población:**

```
[
0:10000
1:10000
2:10000
3:10000
4:10000
5:10000
6:10000
7:10000
8:9357.96
9:10000
]
```

Mejor individuo: 10000.0

Caso #4: Evaluación Mixta para Ver Convergencia

- **Generaciones:** 50
- **Latitud:** 41°
- **Temporada:** Verano
- **Objetivo:** Usar una cantidad reducida de generaciones pero en condiciones menos críticas para ver si el algoritmo converge rápido hacia una solución aceptable.

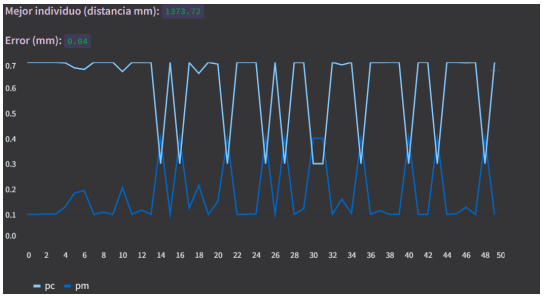

Figura 12. Error durante generaciones Caso #4

Generación 1:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:7149.68
1:1911.76
2:1987.61
3:1948.71
4:9044.1
5:8426.91
6:1699.2
7:622.66
8:9639.58
9:7698.64
]
```

Mejor individuo: 1699.2

Generación 2:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:1699.2
1:1699.2
2:2871.23
3:2321.47
4:1847.99
5:1762.97
6:1699.2
7:1699.2
8:1699.2
9:1699.2
]
```

Mejor individuo: 1699.2

Generación 49:

- **pc =** 0.4
- **pc =** 0.3
- **Población:**

```
[
0:1373.72
1:1373.72
2:1373.72
3:1373.72
4:1373.72
5:1373.72
6:1373.72
7:1373.72
8:1373.72
9:1373.72
]
```

Mejor individuo: 1373.72

Generación 50:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:1617.12
1:1373.72
```

```
2:1516.45
3:910.9
4:1373.72
5:635.11
6:1373.72
7:2207.53
8:1373.72
9:946.1
]
```

Mejor individuo: 1373.72

Caso #5: Prueba Extrema con Iteraciones Elevadas en Condiciones Desafiantes

- **Generaciones:** 200
- **Latitud:** 60°
- **Temporada:** Invierno
- **Objetivo:** En este escenario se incrementa la cantidad de iteraciones para contrarrestar la dificultad que supone una latitud alta en invierno. Es la prueba "sin piedad"para comprobar hasta qué punto se puede optimizar la configuración.

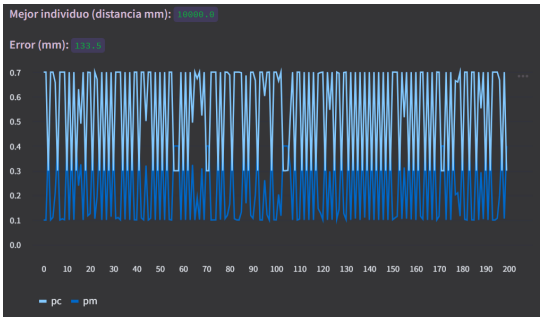

Figura 13. Error durante generaciones Caso #5

Generación 1:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:4520.63
1:2063.31
2:2627.54
3:1580.21
4:435.24
5:8928.19
6:9727.49
7:4318.73
8:1077.17
9:8030.64
]
```

Mejor individuo: 9727.49

Generación 2:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:8658.92
1:8299.91
2:9727.49
3:9727.49
4:9727.49
5:9727.49
6:8928.19
7:9727.49
8:9007.49
9:9727.49
]
```

Mejor individuo: 9727.49

Generación 199:

- **pc =** 0.1058
- **pc =** 0.7
- **Población:**

```
[
0:10000
1:10000
2:9624.42
3:10000
4:10000
5:10000
6:10000
7:10000
8:10000
9:10000
]
```

Mejor individuo: 10000.0

Generación 200:

- **pc =** 0.4
- **pc =** 0.3
- **Población:**

```
[
0:10000
1:10000
2:10000
3:10000
4:10000
5:10000
6:10000
7:10000
8:10000
9:10000
]
```

Mejor individuo: 10000.0

Caso #6: Convergencia Rápida en Condiciones Moderadas

- **Generaciones:** 100
- **Latitud:** 20°
- **Temporada:** Invierno
- **Objetivo:** Aquí se busca ver si, en condiciones no tan extremas en cuanto a la latitud, el algoritmo converge satisfactoriamente en un número medio de generaciones,

lo que podría dar pistas sobre ajustar el algoritmo para entornos menos demandantes.

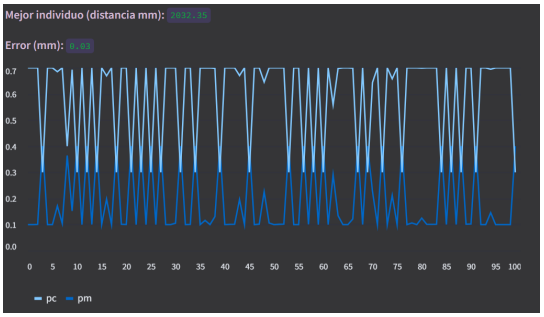

Figura 14. Error durante generaciones Caso #6

Generación 1:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:4253.93
1:7549.76
2:9215.73
3:6681.89
4:1457.27
5:7675.33
6:6561.43
7:5944.82
8:495.34
9:8164.41
]
```

Mejor individuo: 1457.27

Generación 2:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:1457.27
1:1457.27
2:1457.27
3:1457.27
4:2803.54
5:4598.56
6:5944.82
7:4253.93
8:1457.27
9:1457.27
]
```

Generación 99:

- **pc =** 0.1
- **pc =** 0.7
- **Población:**

```
[
0:775.85
1:2032.35
2:2032.35
3:2032.35
4:2032.35
5:2032.35
6:2032.35
7:2032.35
8:2032.35
9:2032.35
]
```

Mejor individuo: 2032.35

Generación 100:

- **pc =** 0.4
- **pc =** 0.3
- **Población:**

```
[
0:2032.35
1:2032.35
2:2032.35
3:2032.35
4:2032.35
5:2032.35
6:2032.35
7:2032.35
8:2032.35
9:2032.35
]
```

Mejor individuo: 2032.35

V. Conclusiones

El desarrollo del taller confirma que el algoritmo genético adaptado, basado en una representación en dos dimensiones, es capaz de optimizar la configuración del panel solar mediante la maximización de la eficiencia energética. La elección de una función objetivo paraboloide, con su característica de poseer un único óptimo global, resulta idónea para este propósito, ya que simplifica el análisis y permite evaluar de forma clara la convergencia del algoritmo. Se concluye que:

Maximización: El problema se plantea como un problema de maximización, dado que se busca alcanzar la mayor eficiencia posible. La función paraboloide, con una cumbre bien definida, facilita la identificación del punto óptimo.

Adaptación del Código: La modificación del código original para trabajar con una representación en tuplas de dos valores (x, y) y el ajuste de los operadores genéticos demuestran la flexibilidad y robustez de la metodología enseñada en clase.

En definitiva, se le recomienda a la empresa Guayepo que diseñe sus paneles solares con 15 grados en el angulo de inclinación y en la orientación para maximizar la eficiencia energética (en base a los resultados de la función matemática escogida).

Referencias

[1] "Declination angle," https://www.pveducation.org/pvcdrom/properties-of-sunlight/declination-angle, accessed: 2025.

[2] “Elevation angle,” https://www.pveducation.org/pvcdrom/properties-of-sunlight/elevation-angle, accessed: 2025.
[3] M. Solar, “How to calculate the minimum distance between pv panels?” https://www.maysunsolar.com/blog-how-to-calculate-the-minimum-distance-between-pv-panels/, accessed: 2025.
[4] E. Eiben, A. and E. Smith, J. *Introduction to Evolutionary Computing*. Springer, 2015.
[5] D. Kim and J. Choi, “Density-based geometric one-class classifier combined with genetic algorithm,” *Mathematical Problems in Engineering*, vol. 2022, pp. 1–18, 04 2022.
[6] F.-J. Chang and L. Chen, “Real-coded genetic algorithm for rule-based flood control reservoir management,” *Water Resources Management*, vol. 12, pp. 185–198, 06 1998.
[7] H. Sosa, S. Villagra, and A. Villagra, “Operadores de mutación en algoritmos genéticos celulares aplicados a problemas continuos,” *Informes Científicos - Técnicos UNPA*, 2013, 1 Becario de Investigación, 2 Docente Investigador UNPA.
[8] M. Srinivas and M. Patnaik, L. “Adaptive probabilities of crossover and mutation in genetic algorithms,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 24, no. 4, pp. 656–667, 1994.
[9] Z. Novikov and E. Sopov, “Development and analysis of adaptive mutation techniques in genetic algorithms,” *ITM Web of Conferences*, vol. 72, p. 05003, 2025.