ISBN: 978-628-95207-4-3. ISSN: 2414-6390. Digital Object Identifier: https://dx.doi.org/10.18687/LACCEI2023.1.1.1262

Siting and Sizing of Distributed Generation Units in Distribution Systems with Genetic Algorithms

Jessica Meza-Zamata, Master[1] , Joaquin Chahuara-Llerena, Bachelor[2], Nestor Gomero-Ostos, Doctor[1], Juan Grados, Doctor[1], Luis Arellán-Yanac, Master[1], Edwin Ramirez-Soto, Master[1] and Juan Lara-Marquez, Doctor[1]

[1]Universidad Nacional del Callao, Perú, jrmezaz@unac.edu.pe, ngomeroo,@unac.edu.pe, jhgradosg@unac.edu.pe, laarellany@unac.edu.pe, eramirezs@unac.edu.pe, jmlaram@unac.edu.pe
[2]Universidad Nacional de San Agustin de Arequipa, Perú, jchahuara@unsa.edu.pe

Abstract– ***This paper presents a genetic algorithm for the location of distributed generation units, using the equations of power flow and energy balance between demanded power and generated power. The genetic algorithm is performed in the open-source programming language Python due to its current importance and multiple libraries for data science. The genetic algorithm for distributed generation localization was applied in a feeder circuit in the city of Arequipa, Peru.***

Keywords-- ***distributed generation, genetic algorithms, power flow, distribution systems.***

Digital Object Identifier: (only for full papers, inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).
DO NOT REMOVE

Ubicación y dimensionamiento de unidades de generación distribuida en sistemas de distribución con algoritmos genéticos

Jessica Meza-Zamata, Master[1] , Joaquin Chahuara-Llerena, Bachelor[2], Nestor Gomero-Ostos, Doctor[1], Juan Grados, Doctor[1], Luis Arellán-Yanac, Master[1], Edwin Ramirez-Soto, Master[1] and Juan Lara-Marquez, Doctor[1]

[1] Universidad Nacional del Callao, Perú, jrmezaz@unac.edu.pe, ngomeroo,@unac.edu.pe, jhgradosg@unac.edu.pe, laarellany@unac.edu.pe, eramirezs@unac.edu.pe, jmlaram@unac.edu.pe

[2] Universidad Nacional de San Agustin de Arequipa, Perú, jchahuara@unsa.edu.pe

***Abstract*– Este trabajo presenta un algoritmo genético para la localización de unidades de generación distribuida, utilizando las ecuaciones de flujo de potencia y balance energético entre potencia demandada y potencia generada. El algoritmo genético se realiza en el lenguaje de programación de código abierto Python debido a su importancia actual y a las múltiples librerías para la ciencia de datos que posee. El algoritmo genético para la localización de generación distribuida se aplicó en un circuito alimentador de la ciudad de Arequipa, Perú.**

***Keywords*-- generación distribuida, algoritmos genéticos, flujo de energía, sistemas de distribución.**

I. Introducción

El desarrollo de nuevas tecnologías y métodos de mejoramiento de redes de distribución, más la experiencia internacional respecto a la inclusión de fuentes de energía renovables y no renovables, frente a la generación de energía eléctrica centralizada, ha logrado que en varios países se adopten dichas tecnologías con su respectiva normativa [1].

El análisis de inclusión de la Generación Distribuida (GD), se realiza mediante la inclusión de aspectos técnicos, económicos y en alguna medida en términos sociales, debido a la aceptación que reciben las energías renovables no convencionales frente a fuentes de energía fósiles [2].

Típicamente los problemas de optimización pueden realizarse con algoritmos evolutivos, especializados en encontrar la mejor solución a un problema determinado, para lo cual se elige de un software de tipo de código abierto llamado Python.

A. Generación Distribuida

La GD generalmente consiste en la generación de energía eléctrica mediante pequeñas fuentes de generación que se instalan cerca de los centros de consumo, por lo que la GD coopera con la generación de energía eléctrica de las grandes centrales eléctricas, haciendo que la demanda energética sea más equilibrada, permitiendo reducir las pérdidas en la red eléctrica [3], al no tener que utilizar redes de transmisión para el transporte de la energía desde las centrales hasta los consumidores finales.

Otro beneficio es el de aumentar la fiabilidad y calidad del sistema eléctrico, debido a que, con varias fuentes de generación de energía, ante una falla en una de las fuentes no implica un grave problema para el sistema de distribución eléctrica. Las potencias de las unidades de microgeneración suelen ser reducidas del orden de los kW (3 – 10 kW) [4].

La microgeneración en pequeñas fuentes de generación eléctrica distribuida por la ciudad, ya sea en edificaciones o lugares públicos, contempla la instalación de energías renovables no convencionales dentro de los cuales se tiene: *i)* Paneles solares, de simple instalación, puede ubicarse en cualquier edificio o área libre, pudiendo lograr autosuficiencia energética en edificaciones, *ii)* Aerogeneradores, son pequeños generadores eólicos pensando para instalarse en fuentes de alumbrado público.

B. Algoritmos Genéticos

La literatura existente sobre ubicación de Algoritmos Genéticos (AG) con métodos numéricos sugiere una amplia variedad de metodologías y criterios para dar solución a este problema. Se suele relacionar con factores técnicos como perdidas y caídas de tensión en las redes y también con factores económicos, agregando penalidades y ponderando factores que permiten cuantificar el problema, mediante funciones objetivo, las cuales se pretende minimizar o maximizar, dependiendo del planteamiento [5].

El método de solución por algoritmos genéticos está inspirado en el proceso de selección natural de las especies biológicas, es decir sobrevive el que está mejor adaptado al medio. Los algoritmos genéticos tienen como partida un conjunto de solución aleatorio, el cual es evaluado por una función de ajuste objetivo o función fitness, esta función busca clasificar las soluciones aleatorias en mejores y peores, para posteriormente mejorarlas siguiendo pautas de la teoría de la evolución. En síntesis, los algoritmos genéticos combinan la aleatoriedad para iniciar un conjunto de soluciones, que luego son dirigidas para buscar el resultado óptimo [6].

Los AG funcionan entre el conjunto de soluciones de un problema llamado fenotipo, y el conjunto de individuos de una población natural, codificando la información en una cadena,

Digital Object Identifier: (only for full papers, inserted by LACCEI).
ISSN, ISBN: (to be inserted by LACCEI).
DO NOT REMOVE

generalmente binaria, llamada cromosoma. Los símbolos que forman las cadenas se llaman genes y cada bit alelo.

Entonces, *i)* El conjunto de alelos (parámetros), se denomina gen, *ii)* Un cromosoma es una cadena de valores y el conjunto de cromosomas es un genotipo, y *iii)* El conjunto solución es denominado fenotipo [7].

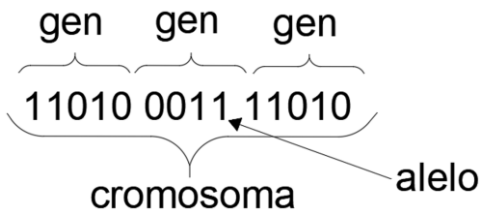

Fig. 1 Esquema de un individuo formado por un numero de bits.

El AG, consiste en una función matemática o rutina de software que toma como entrada un fenotipo inicial, y retorna como salida otro fenotipo, hasta que tenga características que se ajustan mejor a lo que se pretende maximizar o minimizar.

La generación posterior (hijos) a la inicial (padres) está determinada por un conjunto de operadores que combinan y mutan a los primeros miembros.

Operador de cruce, produce nuevos descendientes a partir de las cadenas de bits copiando bits seleccionados de cada padre en los hijos, es decir, combina la información genética de dos individuos (padres) para crear dos individuos (hijos).

Operador de mutación, consiste en la alteración de uno o más valores de genes en un cromosoma, para que el algoritmo llegue a la solución de manera más eficiente.

Función de entrenamiento, es la encargada de definir el criterio para ordenar los fenotipos potenciales y para seleccionarlas probabilísticamente, para incluirlas en la siguiente generación de la población. Aquí se definen criterios los cuales se quieren minimizar o maximizar [8].

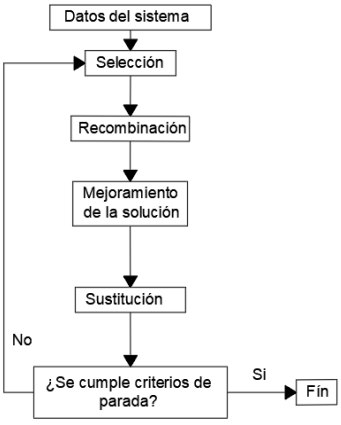

Fig. 2 Bucle de Algoritmo genético con secuencia de operadores genéticos.

El bucle del AG consiste en evaluar la población mediante una función llamada función fitness, función objetivo, función de entrenamiento o función de adaptación.

Repite en las generaciones: i) Elegir "n" individuos de la población original, ii) Aplicar operadores genéticos a los "n" individuos para generar los individuos, y *iii)* Evaluar nuevos individuos según la población de fitness. Reemplazar los peores individuos de la población, por los nuevos individuos creados, hasta cumplir con los criterios de parada [9].

II. METODOLOGÍA

Para describir matemáticamente el problema de ubicación y dimensionamiento de GD, se debe definir la demanda de energía en el sistema de distribución, ecuaciones de restricción para el voltaje en los nodos, ecuaciones de restricción para el rango de potencias de las unidades GD.

$$P_i = \sum_{j=1}^{n} |V_i||V_j||Y_{ij}| \cos\left(\theta_{ij} - \delta_i + \delta_j\right) \quad (1)$$

$$Q_i = \sum_{j=1}^{n} |V_i||V_j||Y_{ij}| \operatorname{sen}\left(\theta_{ij} - \delta_i + \delta_j\right) \quad (2)$$

$$P\,GD\,min < P\,GD < P\,GDmax \quad (3)$$

$$Vmin < Vi < Vmax \quad (4)$$

$$\delta\,min < \delta < \delta\,max \quad (5)$$

Donde:
P_i: es el flujo de potencia activa
Q_i: es el flujo de potencia reactiva.
V_i: es la magnitud de tensión de una barra i.
δ_i: es el ángulo de tensión
Y_{ij}: es la admitancia
P GD: es la potencia activa de una unidad de GD

Las ecuaciones (1) y (2), son las ecuaciones de flujo de carga. Las ecuaciones (3), (4) y (5) representan los límites máximos y mínimas de regulación de tensión [9], [10], ángulo de tensión y potencia de GD en cada nodo.

A. Función Objetivo

La función objetivo tiene datos de entrada y de salida. Los datos de entrada son un vector que contiene la posición y magnitud de unidades de GD, la tensión en barra y el ángulo de tensión en barra: [unidadesGD, magnitud de tensión, Angulo de tensión].

El vector de entrada es una cadena de dígitos binarios (0, 1), se considera 16 bits para cada número del vector de entrada; el número de elementos del vector, considerando que debe haber tres tipos de datos para cada barra del sistema eléctrico, por lo que el número de dígitos para cada vector de entrada es: (# de bits)*(# de barras) [11].

El valor de salida de la función objetivo es la sumatoria de los elementos en términos de potencia activa, la cual por teoría debe ser igual a cero. Se utiliza el valor absoluto, para que en la solución no se consideren números negativos [12].

$$\sum P_{GD} + \sum P_{se} - \sum P_{ij} - \sum P_D \quad (6)$$

Donde:
P GD: es la potencia activa de una unidad GD
P_{se}: es la potencia de las subestaciones del sistema
P_{ij}: son los flujos de potencia
P_D: es la potencia demandada

La ecuación 6 plasmada en la función objetivo es:
return abs(sum(Pse)/100+sum(PGDD)/100-sum(sum(P))-sum(Pd)/100)

El algoritmo buscara la mejor combinación de los datos de entrada para la ecuación anterior (valor de retorno de función objetivo).

El código se desarrolla en Python ya que es un lenguaje de programación muy extendido en el análisis de datos, debido a las múltiples módulos o librerías para el procesamiento de datos, para resolver el problema, el código abarca primero el cálculo de admitancias, para el posterior cálculo de flujo de potencia (las líneas verdes son comentarios) [13].

```
#construccion de matriz de admitancias [Y]
Ybarra=np.zeros((nb,nb),dtype=complex) #matriz de ceros nbxnb

# Elementros de matriz fuera de la diagonal principal
for i in range(nl):
    Ybarra[from_bus[i]-1,to_bus[i]-1]=Ybarra[from_bus[i]-1,to_bus[i]-1]-y[i]
    Ybarra[to_bus[i]-1,from_bus[i]-1]=Ybarra[from_bus[i]-1,to_bus[i]-1]
# Elementos de la diagonal
for i in range(nb):
   for k in range(nl):
      if from_bus[k]==i+1:
         Ybarra[i,i]=Ybarra[i,i]+(y[k])
      elif to_bus[k]==i+1:
         Ybarra[i,i]=Ybarra[i,i]+(y[k])

#  CONDUCTANCI G Y SUSCEPTANCIA B DE ADMITANCIAS Y=G+Bi
G=Ybarra.real
B=Ybarra.imag
```

Los operadores del AG: selección, cruzamiento y mutación se implementan mediante arreglos en tudio (vectores, listas):

```
def selection(pop, scores, k=3):
   #  tudio ón aleatoria
   selection_ix = randint(len(pop))
   for ix in randint(0, len(pop), k-1):
      #  tudio ón n  de el mejor individuo
     if scores[ix] < scores[selection_ix]:
         selection_ix = ix
   return pop[selection_ix]
# operador de cruzamiento
# cruzamiento de dos padres para obtener dos hijos
def crossover(p1, p2, r_cross):
   # los hijos son copias de los padres por defecto
   c1, c2 = p1.copy(), p2.copy()
   #  4tudio4ón4n4 de 4tudio4ón4n4
   if rand() < r_cross:
      #  4tudio4ón de punto de cruzamiento
      pt = randint(1, len(p1)-2)
      # resultado de cruzamiento
      c1 = p1[:pt] + p2[pt:]
      c2 = p2[:pt] + p1[pt:]
   return [c1, c2]

# operador de mutacion
def mutation(bitstring, r_mut):
   for i in range(len(bitstring)):
      #  4tudio4ón de mutacion
      if rand() < r_mut:
         # invertir bit
         bitstring[i] = 1 – bitstring[i]
```

La población inicial para este problema es un vector o lista de dimensión [números de barras * número de bits], ya que los operadores funcionan con cadenas de bits (16 bits para este caso), cada elemento representa la ubicación y dimensionamiento de unidades GD. El rango para unidades GD es [0 – 75] kW.

```
Pop=[]
   for i in range(n_pop):
      a=nb*n_bits*[0]
      pop.append(a)
   #print(pop)
   for j in range(n_pop):
      for i in range(3):
         c=randint(0,nb-1)*n_bits
      pop[j][c:c+n_bits]=list(randint(0,2,n_bits*len([75])))
```

La función objetivo está orientada a minimizar pérdidas en la red, la función objetivo es:

```
def objective(variab):  # pop [lista de binarios]Argumento de función objective
   PGD=variab[0:nb]
```

```
    PGDD=nb*[0]
    num_GD=3 # numero de unidades GD
    for i in range(num_GD):
       PGDD[randint(0,nb)]=PGD[i]
    V=variab[nb:2*nb]
    Theta=variab[2*nb: 3*nb]
    P=list(np.zeros((nb,nb)))
    Q=list(np.zeros((nb,nb)))

  Pd=list(np.array([0,0,0,0,98,0,0,0,0,24.603,24.6,245,24.6,
49.2,98,0,24.5,24.5,24.5,98,680,3228.3,397.72,393,1222.5,79
7.36]))
  Qd=list(np.array([0,0,0,0,19.88,0,0,0,0,4.43,4.43,49.74,4.
43,8.87,19.89,0,4.97,4.97,19.89,421.42,1066.8,42.64,74.5,260
.75,64.93]))

  Pse=nb*[0]
  Pse[0]=12000
  Qse=nb*[0]
  Qse[0]=8000
  largest=2**n_bits

  for i in range(nb):
      #  PGDis=pop_in[i]

     for j in range(nb):

  #P[i][j]=limit_per[i][0]+(P[i][j]/largest)*(limit_per[i][1] -
limit_per[i][0])
       P[i][j]=V[i]*V[j]*(G[i,j]*np.cos(Theta[i]-
Theta[j])+B[i,j]*np.sin(Theta[i]-Theta[j]))
         #Q[i][j]=V[i]*V[j]*(G[i,j]*np.sin(Theta[i]-
Theta[j])-B[i,j]*np.cos(Theta[i]-Theta[j]))+Qse[i]-Qd[i]
    print("ubicacion  de GD:",PGDD)
    #funcion objetivo a minimizar
    return              abs(sum(Pse)/100+sum(PGDD)/100-
sum(sum(P))-sum(Pd)/100)
```

Finalmente, en una función se incluyen los operadores selección, cruzamiento y mutación. Esta función tiene como argumentos de entrada la población inicial, la función objetivo, límites de unidades GD, número de bits, número de iteraciones, número de individuos en la población, porcentaje de cruzamiento y porcentaje de mutación [14],[15].

```
def genetic_algorithm(objective, bounds, n_bits, n_iter,
n_pop, r_cross, r_mut):
    # iniciar población aleatoria de cadena de bits
    #pop = [randint(0, 2, n_bits*len(bounds)).tolist() for _
in range(n_pop)]

    pop=[]
    for i in range(n_pop):
      a=nb*n_bits*[0]
      pop.append(a)
    #print(pop)
    for j in range(n_pop):
      for i in range(3):
        c=randint(0,nb-1)*n_bits
      pop[j][c:c+n_bits]=list(randint(0,2,n_bits*len([75])))
    #print("la  5tudio5ón aleatoria es:",pop_in,"un ind tiene
5tudio5ón: ",len(pop_in[0]))

  # seguir rastro de los mejores individuos
    best, best_eval = 0, (objective(decode(bounds, n_bits,
pop[0])))

  # enumerar generaciones
    for gen in range(n_iter):
      # decodificar población
      decoded = [decode(bounds, n_bits, p)
     for p in pop]
      #print("decoded es",decoded)
      # evaluar todos los candidatos de la población
      scores = [objective(d) for d in decoded]
      # verificación de la nueva mejor población
      for i in range(n_pop):
        if scores[i] < best_eval:
          best, best_eval = pop[i], scores[i]
          print(">%d, new best f(%s) = %f" % (gen,
decoded[i], scores[i]))
      # seleccionar padres
      selected = [selection(pop, scores) for _ in
range(n_pop)]
      # crear la siguiente generación
      children = list()
      for i in range(0, n_pop, 2):
        # get selected parents in pairs
        p1, p2 = selected[i], selected[i+1]
        # crossover and mutation
        for c in crossover(p1, p2, r_cross):
          # mutation
            mutation(c, r_mut)
          # store for next generation
          children.append©
      # replace population
      pop = children
    return [best, best_eval]
```

La salida de este bucle es el mejor individuo y su evaluación según la función objetivo. Algunos argumentos de entrada se definen así:

```
  # numero de iteraciones
  n_iter = 50
  # bits per variable
  # define the population size
  # probabilidad de cruzamiento
```

```
r_cross = 0.9
# probabilidad de mutacion
r_mut = 1.0 / (float(n_bits) * len(bounds))
```

III. Aplicación de la Metodología

El caso de estudio corresponde, al alimentador primario llamado Lambramani de la ciudad de Arequipa, en Perú.

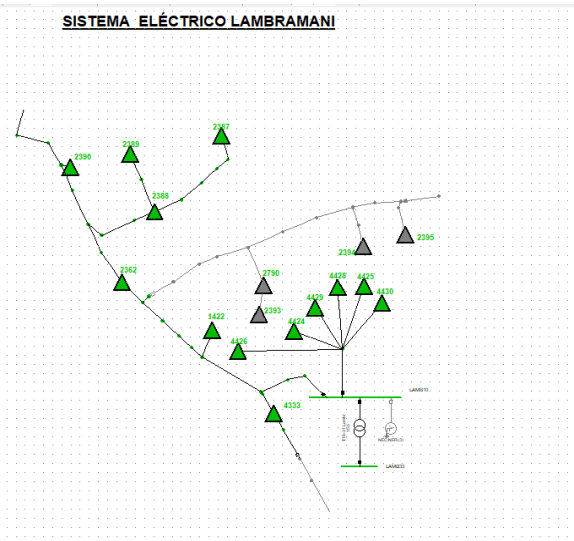

Fig. 3 Sistema Eléctrico de Lambramani.

El circuito de Lambramani se adecuó para ser ejecutado en Python, a fin de obtener un sistema equivalente para ser ejecutado con el algoritmo:

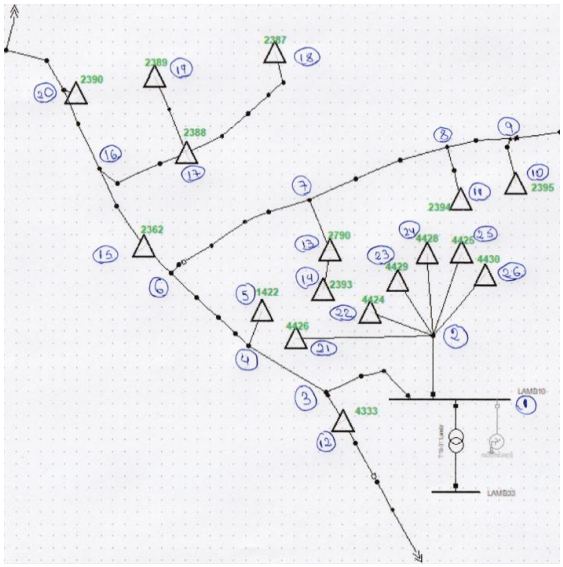

Fig. 4 Circuito de alimentador de Lambramani.

TABLA I
Conexión Entre Nodos y Características de Líneas

DEL NODO	HACIA EL NODO	RESISTENCIA (Ω)	REACTANCIA (Ω)
1	2	0.0098	0.025
2	21	0.1058	0.1158
2	22	0.02984	0.045
2	23	0.0763	0.0475
2	24	0.0501	0.0549
2	25	0.0344	0.0377
2	26	0.093	0.1425
1	3	0.4783	0.0346
3	12	0.0476	0.0224
3	4	0.0218	0.0553
4	5	0.0261	0.0123
4	6	0.0395	0.057
6	7	0.152	0.0864
7	13	0.1215	0.0571
7	8	0.1869	0.0879
8	9	0.0843	0.0396
8	11	0.1722	0.0587
9	10	0.0492	0.0231
6	15	0.0205	0.0216
13	14	0.0522	0.0246
15	16	0.0526	0.0553
16	17	0.1274	0.0569
17	18	0.1471	0.0602
17	19	0.1027	0.0484
16	20	0.0434	0.0456

TABLA II
Potencias de Carga para el Caso de Máxima Demanda

Nodo	P (Kw)	Q (Kvar)
5	98	19.88
10	24.603	4.43
11	24.6	4.43
12	245	49.74
13	24.6	4.43
14	49.2	8.87
15	98	19.89
17	24.5	4.97
18	24.5	4.97
19	24.5	4.97
20	98	19.89
21	680	421.42
22	3228.3	1066.8
23	397.72	42.64
24	393	74.5
25	1222.5	260.75
26	797.36	64.93

IV. Resultados

Un AG es una técnica evolutiva que busca una solución lo suficientemente aceptable, por lo que pueden existir un número de combinaciones que satisfagan la condición de la función objetivo.

El circuito del caso corresponde al circuito del alimentador primario llamado Lambramani. Para el primer caso se considera tres posiciones para unidades GD. Como el algoritmo genético es un método que busca soluciones lo

suficientemente satisfactorias, se ejecutó el algoritmo genético 3 veces y se compara el error para el balance de potencias activas (ecuación 6).

TABLA III
CASO 1: TRES POSICIONES DE UNIDADES GD

Nodo	GD (kW)	GD (kW)	GD (kW)	GD (kW)
1	0.00			7.97
2	0.00			32.43
3	0.00			
4	3.31	61.40		
5	0.00			
6	0.00		68.91	
7	0.00			
8	0.00			
9	65.12			
10	0.00			
11	0.00	31.60		
12	0.00			
13	0.00			
14	0.00			
15	0.00		18.00	
16	0.00			
17	0.00			
18	0.00			31.05
19	0.00	54.54		
20	0.00			
21	0.00			
22	0.00			
23	0.00			
24	33.86		46.76	
25	0.00			
26	0.00			
Error F.O.	0.00004	0.000459	0.000035	0.000057

Siendo la función objetivo un valor cercano a cero, cumple con la condición planteada. Si se considera dos posiciones para ubicar unidades GD, se tiene el siguiente resultado:

TABLA IV
CASO2: DOS POSICIONES DE UNIDADES GD

Nodo	GD (kW)	GD (kW)	GD (kW)	GD kW
1	0.00			
2	0.00			
3	0.00			
4	0.00			
5	0.00			
6	0.00			
7	0.00	42.43		
8	0.00			
9	0.00			
10	0.00			
11	0.00			
12	0.00		28.48	
13	0.00			
14	0.00			
15	52.22			
16	0.00			
17	56.96	63.34		
18	0.00			
19	0.00			
20	0.00			
21	0.00		60.96	
22	0.00			
23	0.00			
24	0.00			
25	0.00			
26	0.00			54.75
				47.63
F.O.	0.00008	0.000004	0.000647	0.000004

TABLA V
CASO3: UNA POSICIÓN DE UNIDADES GD

Nodo	GD (kW)	GD (kW)	GD (kW)	GD (kW)
1	27.28			
2	0.00			
3	0.00			
4	0.00			
5	0.00			
6	0.00			
7	0.00			
8	0.00			
9	0.00			
10	0.00			
11	0.00		59.022	
12	0.00			
13	0.00			
14	0.00			
15	0.00			
16	0.00			
17	0.00			
18	0.00			
19	0.00			
20	0.00	29.74		
21	0.00			
22	0.00			
23	0.00			25.31
24	0.00			
25	0.00			
26	0.00			
FUNCION OBJETIVO	0.000009	0.016007	0.000293	0.000013

V. CONCLUSIONES

El alimentador de Lambramani es un circuito que se encuentra en la ciudad de Arequipa, por lo que presenta ciertas limitaciones respecto a áreas disponibles.

Por las características de radiación solar de Arequipa, se puede implementar algún tipo de grupo de paneles solares del orden de kW siendo factible disponer de varias opciones de ubicaciones, que, para este caso, se considera minimizar las de pérdidas de potencia activa de la red.

Los algoritmos genéticos es una metodología que busca una solución lo suficientemente aceptable, es decir, de entre muchas posibles soluciones busca las que sean óptimamente aceptables.

AGRADECIMIENTO

Los autores agradecen el apoyo otorgado por la Universidad Nacional del Callao y la Universidad Nacional de San Agustín de Arequipa, para el desarrollo de la presente investigación.

REFERENCIAS

[1] V. A. Gómez, C. Hernández, y E. Rivas, "La Influencia de los Niveles de Penetración de la Generación Distribuida en los Mercados Energéticos," Información Tecnológica. Bogotá, vol. 29(1), pp. 117-128, 2018

[2] L. F. Grisales, B. Restrepo, y F. Jaramillo, “UBICACIÒN Y DIMENSIONAMIENTO DE GENERACIÓN DISTRIBUIDA: UNA REVISIÓN, Ciencia e Ingenieria Neogranadina., vol. 27, no. 2, pp. 157-176.
[3] V. H. Gualotuña, “UBICACIÒN ÓPTIMA DE GENERACIÓN DISTRIBUIDA EN SISTEMAS DE DISTRIBUCIÓN UTILIZANDO UN ALGORITMO MULTIOBJETIVO CONSIDERANDO DESPACHO,” Tesis, Universidad de Quito, Quito, 2019.
[4] M. Hernández, “APLICACIÓN DE ALGORITMOS GENÉTICOS PARA LA LOCALIZACIÓN ÓPTIMA DE CAPACITORES EN REDES PRIMARIAS DE DITRIBUCIÓN,’’ proyecto de grado, Universidad de los Andes, 2008.
[5] G. Eulate, E. Rivas, “EL USO DE PYTHON COMO HERRAMIENTA PARA EL ANÁLISIS EN LA INDUSTRIA ELÉCTRICA”.
[6] M. López, R. Gallego, R. Hincapié, “MEJORAMIENTO DEL PERFIL DE TENSIÓN EN SISTEMAS DE DISTRIBUCIÓN USANDO GENERACIÓN DISTRIBUIDA,’’ Scientia et Technica, No 44, Universidad Tecnológica de Pereira ISSN 0122-1701.
[7] G. Candelaria, “METODOLOGÍA GENERAL PARA LA PLANEACIÓN Y OPTIMIZACIÓN DE LAS REDES DE DISTRIBUCIÓN CONSIDERANDO LA GENERACIÓN DISTRIBUIDA,’’ tesis de maestria, Universidad Autónoma del estado de Morelos ,2019.
[8] O. Montoya, C. Ramírez, L. Grisales, “Localización y Dimensionamiento Óptimo de Generadores Distribuidos y Bancos de Condensadores en Sistemas de Distribución,’’ Scientia et Technica, Vol. 23 No 03, Universidad Tecnológica de Pereira, 2018.
[9] A. Gómez, “UBICACIÓN ÓPTIMA DE FUENTES DE GENERACIÓN DISTRIBUIDA EN EL SISTEMA ELÉCTRICO DE DISTRIBUCIÓN,’’ tesis de maestria, Instituto Tecnológico de Ciudad Madero – Tecnológico Nacional de México, 2015.
[10] P. Narváez. J. López – Lezama, E. Velilla. “Ubicación de Generación Distribuida para Minimización de Pérdidas Usando un Algoritmo Genético Híbrido,’’ Información Tecnológica Vol. 26(3), 123 – 132, Medellín – Colombia, 2015.
[11] P. Onofre, “DIMENSIONAMIENTO ÓPTIMO DE GENERACIÓN DISTRIBUIDA EN REDES DE DISTRIBUCIÓN BASADO EN LA TEORÍA DE GRAFOS,’’ Tesis De Grado, Universidad Politécnica Salesiana Sede Quito, 2020.
[12] M. Rubio, “UBICACIÓN Y DIMENSIONAMIENTO DE GENERACIÓN DISTRIBUIDA EN REDES DE MEDIA TENSIÓN EN MERCADOS COMPETITIVOS,’’ tesis de grado para optar grado de magister en Ciencias de la Ingeniería, Pontificia Universidad Catolica de Chile, 2008.
[13] V. Cholota, “IMPACTO DE LA GENERACIÓN DISTRIBUIDA EN REDES DE DISTRIBUCIÓN, APLICACIÓN CENTRAL HIDROELÉCTRICA MIRA,’’ Tesis de grado, Escuela Politécnica Nacional, Quito, 2014.
[14] M. Arias, “DESARROLLO DE UNA HERRAMIENTA INFORMÁTICA PARA LA SIMULACIÓN DE MERCADOS ELÉCTRICOS. APLICACIÓN A LA ASIGNATURA “INGENIANDO UN SISTEMA ELÉCTRICO”., ESCUELA TÉCNICA SUPERIOR DE INGEIEROS DE MINAS Y ENERGÍA, 2016.
[15] “Programación para Ingeniería Eléctrica,’’ Notas del curso de Programación y Lenguajes’’ Facultad de Ingeniería, Universidad de la República, Montevideo – Uruguay, 2015.