



Universidad Nacional Autónoma de México

FACULTAD DE CIENCIAS

REPORTE DE PROYECTO

Optimización de Rutas para la Gestión de Residuos Sólidos con Algoritmos
Genéticos

PROFESOR

M. en C. Oscar Hernández Constantino

AYUDANTES

- Malinali González Lara

PRESENTA

Javier Alejandro Mancera Quiroz, 319274831

Miguel Angel Brito Lievano, 319506330

29 de mayo de 2024

Reporte: Optimización de Rutas para la Gestión de Residuos Sólidos con Algoritmos Genéticos

La gestión eficiente de residuos sólidos es crucial para minimizar los costos de transporte, reducir las emisiones de carbono y garantizar una recolección oportuna. En este proyecto, utilizaremos algoritmos genéticos para optimizar las rutas de recolección de basura en una ciudad.

Descripción del Problema

- **Objetivo:** Minimizar la distancia total recorrida por los camiones de recolección mientras se visitan todos los contenedores de basura.

Datos disponibles:





- Cantidad de camiones.
- Los camiones cargan una cantidad adecuada de residuos.
- Restricciones de tiempo para cada camión (horarios de recolección, ventanas de tiempo).
- Mapa de la ciudad con calles y restricciones de tráfico.
- Ubicaciones de los contenedores de basura.
- Demanda de residuos en cada contenedor (cantidad de basura generada).

Representación de Soluciones

- Cada solución estará representada como un conjunto de permutaciones.
- Cada permutación representará el recorrido de un camión.

Así, una solución se ve de la forma:

$[(a,b,...c), (d,e,...f), (g,h,...i), (j,k,...l)]$

Camión	Recorrido
	(a,b,...c)
	(d,e,...f)
	(g,h,...i)
	(j,k,...l)

Función Objetivo

- La función objetivo evaluará la calidad de una ruta:

Minimizar la distancia total recorrida.

Cumplir con las restricciones de tiempo.

Equilibrar la carga de los camiones.

Modelar la variabilidad de la cantidad de basura generada (nuevo objetivo).

- Por recorrido, se realiza
 - 1.Evaluación de los pesos de cada arista (tráfico de cada calle)
 - 2.Evaluación de la cantidad de residuos que se recogió.
 - 3.Evaluación de tiempo del recorrido
- Por solución:
 - 4.Sumas de todas las evaluaciones hechas para cada camión.

Se aplicarán penalizaciones grandes por estar fuera de horario en la función de evaluación.

Un ejemplar de prueba está compuesto de:

Número de camiones disponibles y el valor óptimo conocido (si está disponible). El tipo de problema que intentamos resolver es el CVRP, debemos encontrar rutas óptimas para un conjunto de vehículos que deben entregar bienes a un conjunto de clientes desde un depósito central.

Información sobre los Nodos:

- Cada nodo representa un cliente o una ubicación.
- Las coordenadas de los nodos (generalmente en un plano 2D) indican dónde se encuentran físicamente.
- Por ejemplo, el nodo 1 puede estar en las coordenadas (76, 75).

Demanda de los Nodos:

- Cada nodo tiene una demanda asociada (por ejemplo, la cantidad de bienes que debe entregarse).
- La sección "DEMAND_SECTION" enumera las demandas de cada nodo.

Capacidad de los Vehículos:

- Se especifica la capacidad máxima de los vehículos (por ejemplo, 100 unidades).

- Los vehículos no pueden exceder esta capacidad al realizar entregas.

Depósito (Depot):

- El depósito es el punto de partida y regreso para los vehículos.
- Generalmente se representa como el nodo 1.

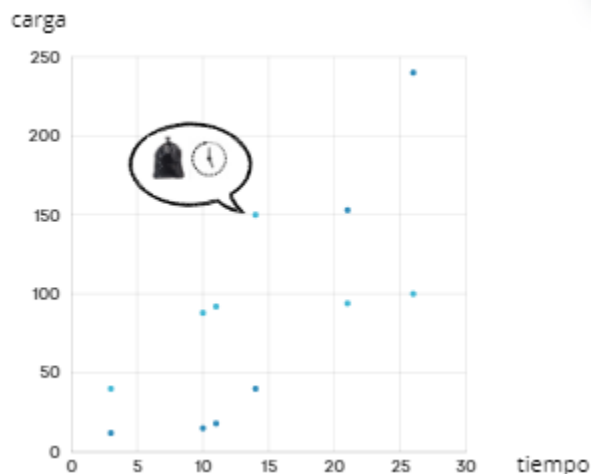
NSGA-II

Con este algoritmo buscamos encontrar las soluciones con un mínimo tiempo de recolección y mayor peso cargado para cada camión. Esto requiere de un algoritmo genético que pueda abordar problemas multiobjetivo como NSGA-II. A continuación explicaremos cómo funciona NSGA-II en nuestro algoritmo.

Rango

Existe un rango de soluciones con diferentes valores de tiempo y carga. Algunos optimizarán más la carga mientras que otros optimizarán más el tiempo. Buscamos que las soluciones óptimas realicen una optimización de ambos rubros. Por lo tanto, si representamos las soluciones en una gráfica con respecto al tiempo y carga, podemos identificar las soluciones dentro de una línea en la gráfica, en el extremo de donde se encuentran los puntos de las soluciones.

En el NSGA-II, se utilizan además de selección por torneo, cruza y mutación, procesos como non-dominating sorting y crowd distance sorting, que se explicarán a continuación.

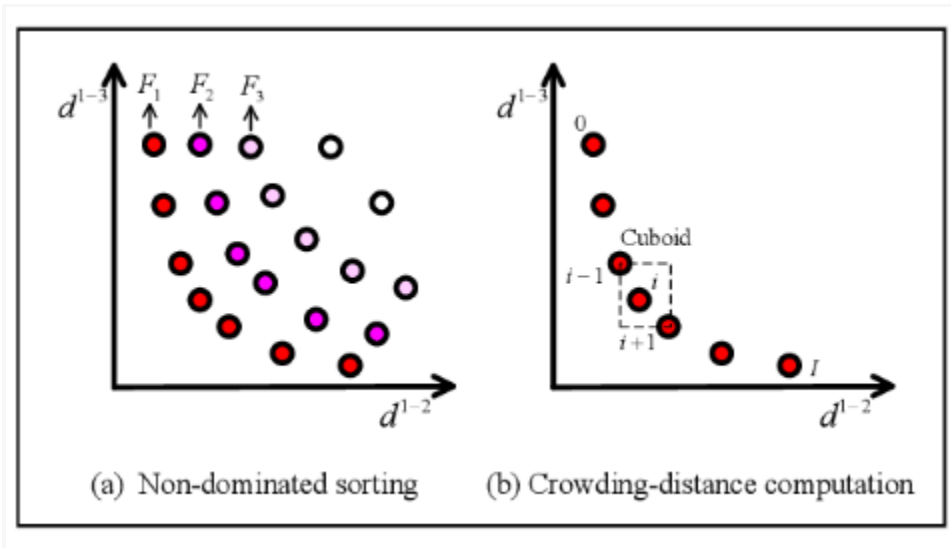


Representación de rango de soluciones

Non-dominating sorting

En el proceso de optimización multiobjetivo, el ordenamiento no dominado tiene como objetivo encontrar soluciones que optimicen múltiples objetivos conflictivos simultáneamente. La relación de dominancia establece que un individuo domina a otro si es mejor o igual en todos los objetivos y estrictamente mejor en al menos uno. Por ejemplo, si una solución tiene mejores valores en ambos Objetivo 1 (F_1) y Objetivo 2 (F_2) que otra solución, domina a la última.

El primer frente consta de individuos no dominados (aquellos que no son dominados por ningún otro). Los frentes subsiguientes contienen soluciones progresivamente menos óptimas. Cada individuo recibe un rango basado en el frente al que pertenece. El frente 1 tiene rango 1, el frente 2 tiene rango 2, etc.

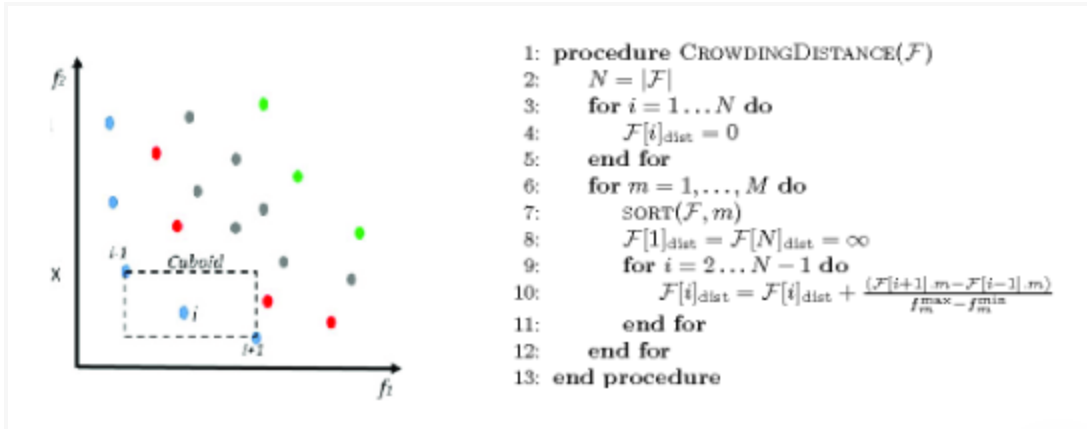


El ordenamiento no dominado ayuda a identificar compensaciones y proporciona un conjunto diverso de soluciones para los tomadores de decisiones.

Crowding distance:

El objetivo de calcular la crowding distance es decidir que soluciones pasarán a la siguiente generación después del criterio de los rangos.

Para cada individuo, calcula la distancia de agrupamiento considerando las distancias entre individuos vecinos en el mismo frente para cada objetivo. Los individuos con mayores distancias de agrupamiento tienen prioridad durante la selección.



La distancia de agrupamiento asegura que las soluciones estén bien distribuidas en el frente de Pareto, evitando la agrupación.

Selección por Torneo

En el proceso de Algoritmo Genético (AG) con Crowding Distance, primero se seleccionan dos padres aleatoriamente de la población actual (PT). Luego, se comparan sus rangos: el padre con un rango superior tiene más posibilidades de reproducirse. Si los rangos son iguales, se utiliza la distancia de agrupamiento (crowding distance) para decidir qué padre contribuirá a la descendencia.

Cruza

- Después de seleccionar los padres, se realiza el cruce.
- El cruce mezcla los genes de los individuos elegidos. Los genes que determinan el tiempo requerido y la cantidad de residuo cargado se combinan para formar el ADN de los hijos.
- La mitad de los genes proviene de un padre y la otra mitad del otro.
- Los hijos resultantes se agregan a la población de descendencia.

Durante el proceso, pueden ocurrir mutaciones aleatorias. Por ejemplo, un gen que influye en la cantidad de carga podría aumentar o disminuir en un porcentaje aleatorio.

Implementación

Modelado de Demanda de Residuos:

- Descripción:
 - En lugar de considerar una demanda uniforme para todos los contenedores, modelaremos la variabilidad de la cantidad de basura generada en diferentes áreas de la ciudad.
- Implementación:

1. Recuperar datos, de una representación de un mapa, que nos servirán para establecer los datos de entrada para el problema. Como los nodos de una gráfica, la cantidad de nodos, la cantidad mínima de camiones que se utilizarán y la capacidad máxima que tendrá cada camión para la carga de residuos.
2. Para evaluar cada solución, tomaremos en cuenta el recorrido que hizo cada camión, en la que se obtendrá la distancia de acuerdo al tráfico que hubo, la carga total de residuos, que se penalizará si sobrepasa el límite y el tiempo en que se finalizó todos los recorridos, que también se penalizará si se sobrepasa de acuerdo al horario establecido.
 - $f(x) = \text{Distancia} + \text{Carga} + \text{Tiempo}$

Restricciones de Tiempo y Ventanas de Tiempo:

- Descripción:
 - Consideraremos las ventanas de tiempo en las que los camiones de recolección pueden operar. Algunos contenedores solo pueden ser recogidos en ciertos horarios.
- Implementación:
 1. Definir ventanas de tiempo para cada contenedor (inicio y fin). Por ejemplo:
 - Contenedor A: De 8:00 a 10:00.
 - Contenedor B: De 9:30 a 11:30.
 2. Asegurarse de que las rutas generadas cumplan con estas restricciones.
 3. Se penalizarán las soluciones que no respeten las ventanas de tiempo en la función objetivo.

Para implementar el algoritmo NSGA-II en nuestro problema de optimización de rutas de camiones, usamos los siguientes componentes. A continuación el pseudocódigo de los componentes más importantes:

Título: ordenamiento_no_determinado

Descripción: Esta función implementa un algoritmo de ordenamiento no determinado para dividir una población en diferentes rangos basados en la dominancia de los individuos. Utiliza el concepto de dominancia para clasificar los individuos en rangos.

Función ordenamiento_no_determinado(población):

i = 0

Mientras i < longitud(población):

 dominados = []

 j = 0

 Mientras j < longitud(población):

 Si población[i] != población[j] entonces

Si (población[i].evaluacion_trayecto \geq población[j].evaluacion_trayecto y población[i].evaluacion_carga \geq población[j].evaluacion_carga y población[i].evaluacion_tiempo \geq población[j].evaluacion_tiempo) y (población[i].evaluacion_trayecto > población[j].evaluacion_trayecto o población[i].evaluacion_carga > población[j].evaluacion_carga o población[i].evaluacion_tiempo > población[j].evaluacion_tiempo) entonces

Agregar j a dominados

dominación = lista(población[i].dominación)

población[i] = reemplazar(población[i], dominación, [dominación[0], dominados])

dominación = lista(población[j].dominación)

contador_dominaciones = dominación[0] + 1

población[j] = reemplazar(población[j], dominación, [contador_dominaciones, dominación[1]])

termina si

termina si

j += 1

termina si

i += 1

rango_k = []

cont_ind = 0

i = 0

Mientras i < longitud(población):

dominación = lista(población[i].dominación)

Si dominación[0] == 0 entonces

Agregar i a rango_k

cont_ind += 1

termina si

i += 1

termina si

rangos = []

rangos.agregar(rango_k)

crea_rango = Verdadero

iter = 0

Mientras crea_rango y iter < 100 entonces

rango_n = []

i = 0

Mientras i < longitud(población) entonces

Para cada j en rango_k entonces

Si población[i] != población[j] entonces

dominación_j = lista(población[j].dominación)

Si dominación_j[1] no es Nulo y i está en dominación_j[1] entonces

dominación_i = lista(población[i].dominación)

contador_dominaciones = dominación_i[0] - 1

población[i] = reemplazar(población[i], dominación, [contador_dominaciones, dominación_i[1]])


```

        Si contador_dominaciones == 0 entonces
            Agregar i a rango_n
            cont_ind += 1
        termina si
    termina si
termina para
i += 1
termina mientras
rangos.agregar(rango_n)
Si cont_ind == longitud(población) entonces
    Romper
termina si
rango_k = rango_n
iter += 1
termina mientras
rangos = limpiar_lista(rangos)

Devolver rangos

```

Título: evaluar_recorrido

Descripción: Calcula el tráfico y el número de residuos para un recorrido dado.

Pseudocódigo:

less

Copiar código

Función evaluar_recorrido(trayecto):

```

    trafico = 0
    num_residuos = 0    terminar mientras

    i = 1
    Mientras i < longitud(trayecto) entonces
        trafico += distancia_euclidiana(trayecto[i-1][0], trayecto[i][0])
        num_residuos += trayecto[i][1]
    i += 1
    Devolver trafico, num_residuos

```

Título: evaluar_camion

Descripción: Evalúa un camión en función de su tráfico, carga, y tiempo de finalización.

Pseudocódigo:

Función evaluar_camion(camion, capacidad, penalizacion=1000):

```

    total_de_trafico, total_residuos = evaluar_recorrido(camion[0])
    evaluacion = total_de_trafico
    evaluacion_trafico = total_de_trafico
    evaluacion_carga = total_residuos

```

```
Si total_residuos > capacidad entonces
    evaluacion += evaluacion * penalizacion
    evaluacion_carga += evaluacion_carga * penalizacion
terminar si
```

```
horario = camion[1]
hora_de_termino = camion[2]
hora_dt = convertir_a_time(hora_de_termino)
evaluacion_tiempo = 1
```

```
Si hora_dt > horario[1] entonces
    evaluacion += evaluacion * penalizacion
    evaluacion_tiempo += evaluacion_tiempo * penalizacion
terminar si
```

```
Devolver evaluacion, evaluacion_trafico, evaluacion_carga, evaluacion_tiempo
```

Título: evaluar_individuo

Descripción: Evalúa un individuo en función de su tráfico, carga y tiempo de finalización.

Pseudocódigo:

Función evaluar_individuo(individuo, capacidad):

```
    eval_ind = 0
    eval_trafico = 0
    eval_carga = 0
    eval_tiempo = 0
```

```
    Para cada camion en individuo[0] entonces
        eval, eval_Tr, eval_C, eval_Ti = evaluar_camion(camion, capacidad)
        eval_ind += eval
        eval_trafico += eval_Tr
        eval_carga += eval_C
        eval_tiempo += eval_Ti
    terminar para
```

```
    individuo = reemplazar(individuo, evaluacion=eval_ind,
    evaluacion_trayecto=eval_trafico, evaluacion_carga=eval_carga,
    evaluacion_tiempo=eval_tiempo)
```

```
    Devolver individuo
```

Título: evaluar_poblacion

· Descripción: Evalúa cada individuo en una población en función de su tráfico, carga y tiempo de finalización.

Pseudocódigo:

Función evaluar_poblacion(poblacion, capacidad):

```
    j = 0
    Para cada i en poblacion entonces
        poblacion[j] = evaluar_individuo(i, capacidad)
```

```
j += 1
terminar para
```

Título: genera_poblacion

Descripción: Genera una población de soluciones con representación de conjuntos de permutaciones, donde cada solución consiste en trayectos de camiones, sus horarios y la hora de finalización de su trabajo.

Pseudocódigo:

Función genera_poblacion(num_nodos, num_camiones, coordenadas, horario):

```
poblacion = []
deposito = coordenadas[0]
i = 0
Mientras i < 100 entonces
    coordenadas_agregadas = []
    solucion = []
    periodos = dividir_horas(num_camiones, horario)
    nodos = 1
    camion = 1
    Mientras camion <= num_camiones entonces
        dimension = num_nodos-1
        datos_camion = []
        trayecto = []
        num_contenedores = 0
        trayecto.append(deposito)
        coordenadas_agregadas.append(deposito)
        Si dimension > 0 entonces
            num_contenedores = generar_entero_aleatorio(1, dimension)
        Fin si
        contenedores = 1
        Mientras contenedores <= num_contenedores entonces
            coordenada = seleccionar_aleatoriamente(coordenadas)
            Si coordenada no está en coordenadas_agregadas entonces
                trayecto.append(coordenada)
                coordenadas_agregadas.append(coordenada)
                nodos += 1
            Fin si
            contenedores += 1
        Fin mientras
        Si camion es igual a num_camiones y nodos es menor que num_nodos entonces
            coordenadas_faltantes = obtener_elementos_no_agregados(coordenadas,
            coordenadas_agregadas)
            Para cada coordenada en coordenadas_faltantes entonces
                trayecto.append(coordenada)
                nodos += 1
            Fin para
        Fin si
        trayecto.append(deposito)
```

```

datos_camion.append(trayecto)
datos_camion.append(periodos[camion-1])
hora_que_finalizo = generar_hora_aleatoria(periodos[camion-1], horario[1])
datos_camion.append(hora_que_finalizo)
solucion.append(datos_camion)
dimension = dimension - contenedores
camion += 1
Fin mientras
individuo = crear_individuo(solucion, 0, 0, 0, 0, [0, Ninguno], 0)
poblacion.append(individuo)
i += 1
Fin mientras
Devolver poblacion

```

Pseudocódigo:

Función cruza_de_permutaciones(sol1, sol2, pc):

```

trayecto_de_sol1 = []
trayecto_de_sol2 = []
sol_hijo1 = []
sol_hijo2 = []
trayecto_de_h1 = []
trayecto_de_h2 = []

```

```

Para cada camion en sol1 entonces
Para cada contenedor en trayecto de camion[0] entonces
Si contenedor no es el deposito entonces
    agregar contenedor a trayecto_de_sol1
Fin si
Fin para
Fin para

```

```

Para cada camion en sol2 entonces
Para cada contenedor en trayecto de camion[0] entonces
Si contenedor no es el deposito entonces
    agregar contenedor a trayecto_de_sol2
Fin si
Fin para
Fin para

```

```

tamanio = longitud(trayecto_de_sol1)
indice_comienzo = generar_entero_aleatorio(0, tamanio)
indice_final = generar_entero_aleatorio(indice_comienzo, tamanio)
indices_no_agregados = []
ind = 0
Mientras ind < tamanio entonces
Si ind es mayor o igual a indice_comienzo y ind es menor o igual a indice_final entonces

```

Si generar_numero_aleatorio() es menor o igual a pc entonces
 agregar trayecto_de_sol1[ind] a trayecto_de_h1
 agregar trayecto_de_sol2[ind] a trayecto_de_h2

Fin si

Sino

agregar Nulo a trayecto_de_h1

agregar Nulo a trayecto_de_h2

agregar ind a indices_no_agregados

Fin si

ind += 1

Fin mientras

Para cada indice en indices_no_agregados entonces

Si trayecto_de_h1[indice] es Nulo y trayecto_de_h2[indice] es Nulo entonces

ind = 0

Mientras ind < longitud(trayecto_de_sol1) entonces

 Si trayecto_de_sol2[ind] no está en trayecto_de_h1 entonces

 agregar trayecto_de_sol2[ind] a trayecto_de_h1

 Fin si

 ind += 1

Fin mientras

ind = 0

Mientras ind < longitud(trayecto_de_sol1) entonces

 Si trayecto_de_sol1[ind] no está en trayecto_de_h2 entonces

 agregar trayecto_de_sol1[ind] a trayecto_de_h2

 Fin si

 ind += 1

Fin mientras

Fin si

Fin para

num_camiones = longitud(sol1)

num_nodos = longitud(trayecto_de_h1) + 1

sol_hijo1 = []

camion = 1

nodos = 1

Mientras camion <= num_camiones entonces

dimension = longitud(trayecto_de_h1)-1

datos_camion = []

trayecto_h1 = []

trayecto_h1.append(deposito)

Si dimension > 0 entonces

num_contenedores = generar_entero_aleatorio(1, dimension)

Fin si

contenedores = 1

Mientras contenedores < num_contenedores entonces

```

    contenedor = trayecto_de_h1[0]
    trayecto_h1.append(contenedor)
    remover contenedor de trayecto_de_h1
    contenedores += 1
Fin mientras
Si camion es igual a num_camiones y nodos es menor que num_nodos entonces
    contenedores_faltantes = obtener_elementos_no_agregados(trayecto_de_h1,
trayecto_h1)
    Para cada contenedor en contenedores_faltantes entonces
        trayecto_h1.append(contenedor)
        remover contenedor de trayecto_de_h1
        nodos += 1
Fin para
Fin si
trayecto_h1.append(deposito)
datos_camion.append(trayecto_h1)
datos_camion.append(sol1[camion-1][1])
hora_que_finalizo = generar_hora_aleatoria(sol1[camion-1][1], horario[1])
datos_camion.append(hora_que_finalizo)
sol_hijo1.append(datos_camion)
dimension = dimension - contenedores
camion += 1
Fin mientras

```

```

ind1 = crear_individuo(sol_hijo1, 0, 0, 0, 0, [0, Ninguno], 0)

```

```

num_camiones = longitud(sol2)
num_nodos = longitud(trayecto_de_h2) + 1
sol_hijo2 = []
camion = 1
nodos = 1
Mientras camion <= num_camiones entonces
    dimension = longitud(trayecto_de_h2)-1
    datos_camion = []
    trayecto_h2 = []
    trayecto_h2.append(deposito)
    Si dimension > 0 entonces
        num_contenedores = generar_entero_aleatorio(1, dimension)
    Fin si
    contenedores = 1
    Mientras contenedores < num_contenedores entonces
        contenedor = trayecto_de_h2[0]
        trayecto_h2.append(contenedor)
        remover contenedor de trayecto_de_h2
        contenedores += 1
    Fin mientras

```

```

    Si camion es igual a num_camiones y nodos es menor que num_nodos entonces
        contenedores_faltantes = obtener_elementos_no_agregados(trayecto_de_h2,
trayecto_h2)
        Para cada contenedor en contenedores_faltantes entonces
            trayecto_h2.append(contenedor)
            remover contenedor de trayecto_de_h2
            nodos += 1
        Fin para
    Fin si
    trayecto_h2.append(deposito)
    datos_camion.append(trayecto_h2)
    datos_camion.append(sol2[camion-1][1])
    hora_que_finalizo = generar_hora_aleatoria(sol2[camion-1][1], horario[1])
    datos_camion.append(hora_que_finalizo)
    sol_hijo2.append(datos_camion)
    dimension = dimension - contenedores
    camion += 1
    Fin mientras

    ind2 = crear_individuo(sol_hijo2, 0, 0, 0, 0, [0, Ninguno], 0)

    Devolver ind1, ind2

```

Título: mutacion

Descripción: Realiza la mutación de un individuo, con una probabilidad determinada, alterando la secuencia de contenedores en el trayecto de los camiones.

Pseudocódigo:

Función mutacion(solucion, pm):

```

    trayecto_de_sol = []
    Para cada camion en solucion entonces
        Para cada contenedor en trayecto de camion[0] entonces
            Si contenedor no es el deposito entonces
                agregar contenedor a trayecto_de_sol
        Fin si
    Fin para
    Fin para

    i = 0
    Mientras i < longitud(trayecto_de_sol) entonces
        Si generar_numero_aleatorio() es menor o igual a pm entonces
            temp = trayecto_de_sol[i]
            pos_random = generar_entero_aleatorio(i, longitud(trayecto_de_sol)-1)

```

```

trayecto_de_sol[i] = trayecto_de_sol[pos_random]
trayecto_de_sol[pos_random] = temp
Fin si
i += 1
Fin mientras

deposito = solucion[0][0][0]
camion = 0
ind = 0
Mientras camion < longitud(solucion) entonces
    i = 0
    Mientras i < longitud(solucion[camion][0]) entonces
        Si solucion[camion][0][i] no es el deposito entonces
            solucion[camion][0][i] = trayecto_de_sol[ind]
            ind += 1
    Fin si
    i += 1
Fin mientras
camion += 1

mut = Individuo(solucion, 0, 0, 0, 0, [0, Ninguno], 0)

Devolver mut

```

Título: ordenamiento_por_distancia_de_aglomeracion

Descripción: Realiza el ordenamiento de una población por distancia de aglomeración, utilizando el método NSGA-II.

Pseudocódigo:

Función ordenamiento_por_distancia_de_aglomeracion(poblacion, rangos, capacidad):

```

    indice_pob = 0
    Para cada rango en rangos entonces
        rango_de_poblacion = []
        indice = indice_pob
        Para cada ind en rango entonces
            rango_de_poblacion.append(poblacion[indice])
            indice += 1
        Fin para
        rango_de_poblacion = ordenar(rango_de_poblacion, por evaluar_individuo(individuo,
capacidad).evaluacion)
        rango_de_poblacion[0] = rango_de_poblacion[0]._reemplazar(distancia=infinito)
        rango_de_poblacion[longitud(rango_de_poblacion)-1] =
rango_de_poblacion[longitud(rango_de_poblacion)-1]._reemplazar(distancia=infinito)
        ind = 0
        Mientras ind < longitud(rango_de_poblacion) entonces
            Si rango_de_poblacion[ind] no es igual a rango_de_poblacion[0] y
rango_de_poblacion[ind] no es igual a rango_de_poblacion[longitud(rango_de_poblacion)-1]
entonces

```



```

        distancia_ind = poblacion[ind].distancia
        Si rango_de_poblacion[longitud(rango_de_poblacion)-1].evaluacion -
rango_de_poblacion[0].evaluacion != 0 entonces
            rango_de_poblacion[ind] =
rango_de_poblacion[ind].reemplazar(distancia=distancia_ind+(rango_de_poblacion[ind+1].eva
luacion -
rango_de_poblacion[ind-1].evaluacion)/(rango_de_poblacion[longitud(rango_de_poblacion)-1].e
valuacion - rango_de_poblacion[0].evaluacion))
        Fin si
    Fin si
    ind += 1
Fin mientras
rango_de_poblacion = ordenar(rango_de_poblacion, por distancia, reversa=Verdadero)
Para cada individuo en rango_de_poblacion entonces
    poblacion[indice_pob] = individuo
    indice_pob += 1
Fin para

Devolver poblacion

```

Título: alg_NSGA2

Descripción: Implementa el algoritmo NSGA-II para la optimización multiobjetivo de un problema de rutas de vehículos.

Pseudocódigo:

Función alg_NSGA2(num_nodos, num_camiones, coordenadas, horario, capacidad):

```

    mejor_solucion = Individuo(Ninguno, 0, 0, 0, 0, [0, Ninguno], 0)

```

```

    peor_solucion = Individuo(Ninguno, 0, 0, 0, 0, [0, Ninguno], 0)

```

```

    poblacion = genera_poblacion(num_nodos, num_camiones, coordenadas, horario)

```

```

    gen = 0

```

```

    Mientras gen < 100 entonces

```

```

        imprimir("Generación = " + (gen+1))

```

```

        iter = 0

```

```

        Mientras iter < 50 entonces

```

```

            evaluar_poblacion(poblacion, capacidad)

```

```

            padre1, padre2 = seleccion_por_torneo(poblacion, capacidad)

```

```

            hijo1, hijo2 = cruza_de_permutaciones(poblacion[padre1].solucion,
poblacion[padre2].solucion, generar_decimal_aleatorio(0.6, 0.9))

```

```

            hijo1 = mutacion(hijo1.solucion, generar_decimal_aleatorio(0.01, 0.1))

```

```

            hijo2 = mutacion(hijo2.solucion, generar_decimal_aleatorio(0.01, 0.1))

```

```

            agregar poblacion.append(hijo1)

```

```

            agregar poblacion.append(hijo2)

```

```

            iter += 1

```

```

    tamaño_poblacion = longitud(poblacion)

```

```

evaluar_poblacion(poblacion, capacidad)
orden_indices = ordenamiento_no_determinado(poblacion)
copia_poblacion = poblacion
ind = 0
Para cada rango en orden_indices entonces
Para cada indice en rango entonces
    poblacion[ind] = copia_poblacion[indice]
    ind += 1
Fin para
poblacion = ordenamiento_por_distancia_de_aglomeracion(poblacion, orden_indices,
capacidad)
copia_poblacion = poblacion.copiar()
ind = 0
Mientras ind < tamaño_poblacion entonces
Si ind >= tamaño_poblacion/2 entonces
    remover poblacion.remove(copia_poblacion[ind])
Fin si
ind += 1

mejor_solucion_por_gen = poblacion[0]
peor_solucion_por_gen = poblacion[longitud(poblacion)-1]
Si gen es igual a 1 entonces
    mejor_solucion = mejor_solucion_por_gen
Si mejor_solucion_por_gen.evaluacion < mejor_solucion.evaluacion y
mejor_solucion_por_gen.evaluacion < peor_solucion_por_gen.evaluacion entonces
    mejor_solucion = mejor_solucion_por_gen
Si peor_solucion_por_gen.evaluacion < mejor_solucion.evaluacion y
peor_solucion_por_gen.evaluacion < mejor_solucion_por_gen.evaluacion entonces
    mejor_solucion = peor_solucion_por_gen
Si gen es igual a 1 entonces
    peor_solucion = peor_solucion_por_gen
Si mejor_solucion_por_gen.evaluacion > peor_solucion.evaluacion y
mejor_solucion_por_gen.evaluacion > peor_solucion_por_gen.evaluacion entonces
    peor_solucion = mejor_solucion_por_gen
Si peor_solucion_por_gen.evaluacion > peor_solucion.evaluacion y
peor_solucion_por_gen.evaluacion > mejor_solucion_por_gen.evaluacion entonces
    peor_solucion = peor_solucion_por_gen
Incrementar gen

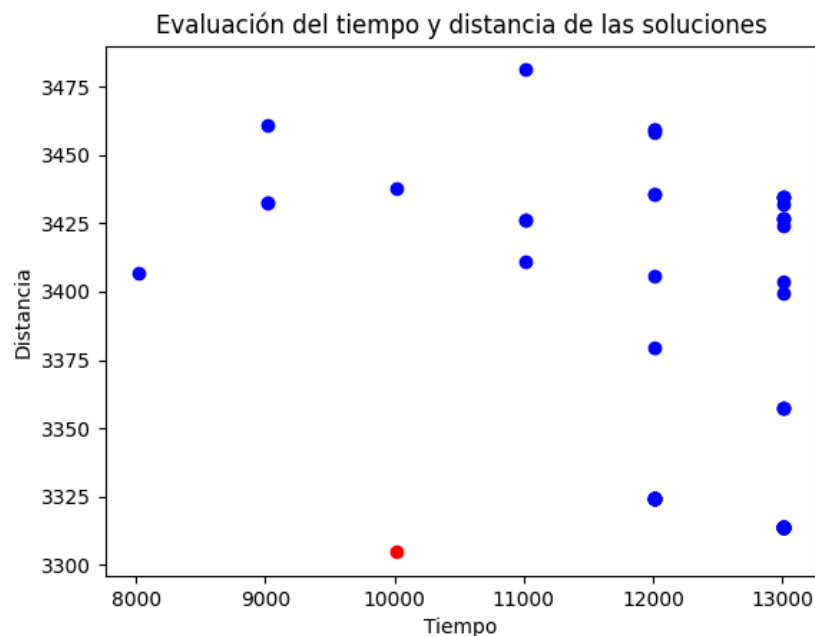
Devolver mejor_solucion

```

Resultados

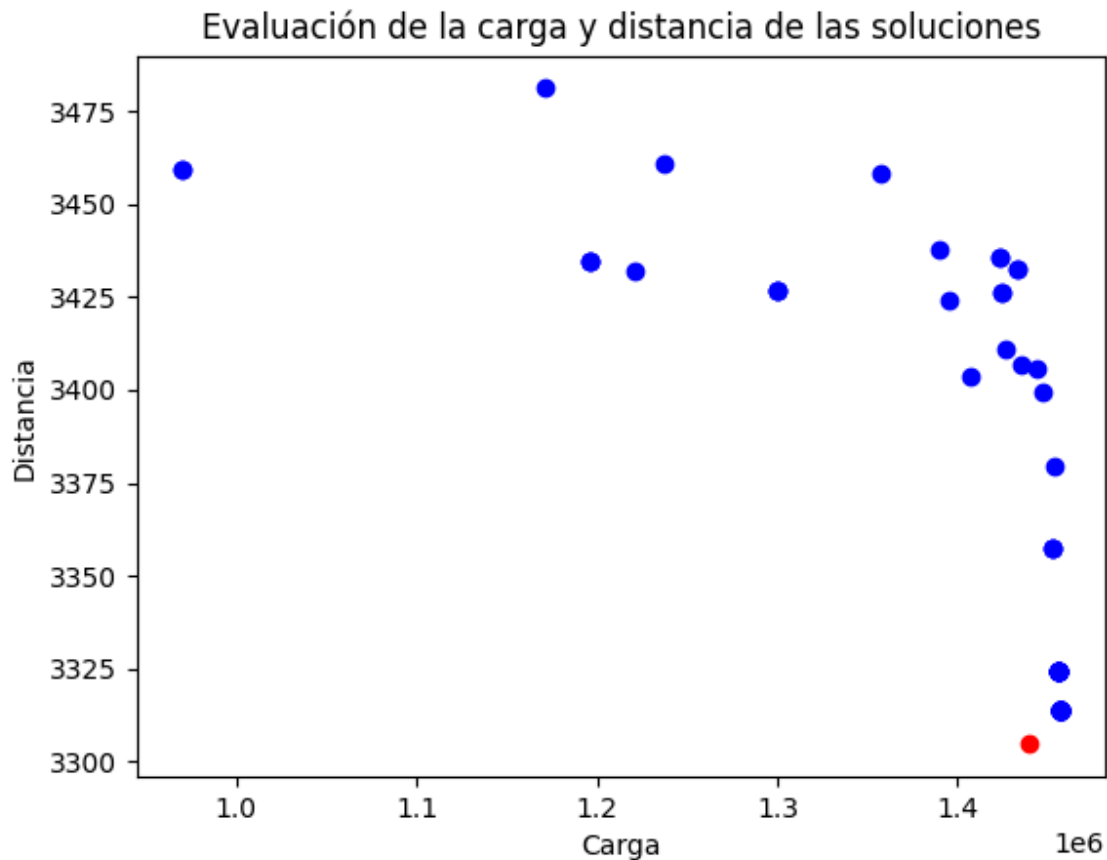
A continuación, presentamos los resultados del algoritmo en los cuales se evalúan las rutas soluciones en cuanto a su tiempo, distancia y carga. Estos resultados corresponden a uno de los ejemplares en los que se aplicó el algoritmo.

Esta gráfica representa una evaluación del tiempo con respecto a la distancia en las soluciones, podemos ver que la ruta seleccionada está en rojo. Esta mejor solución ofrece domina a las demás en cuanto a distancia y ofrece un buen compromiso en el tiempo requerido.



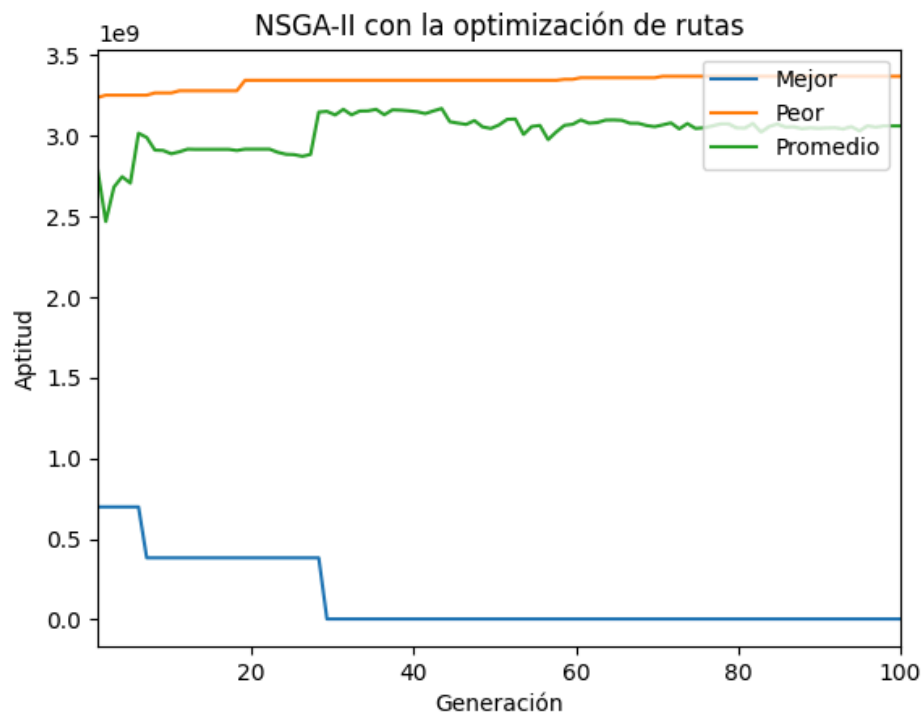
Por otra parte, evaluando la distancia con respecto a la carga, podemos ver que las soluciones candidatas de la última generación después de haber corrido el NSGA-II maximizan satisfactoriamente la cantidad de carga de los camiones. La solución elegida domina a las demás soluciones en cuanto a carga y también es la que menor

distancia tiene.

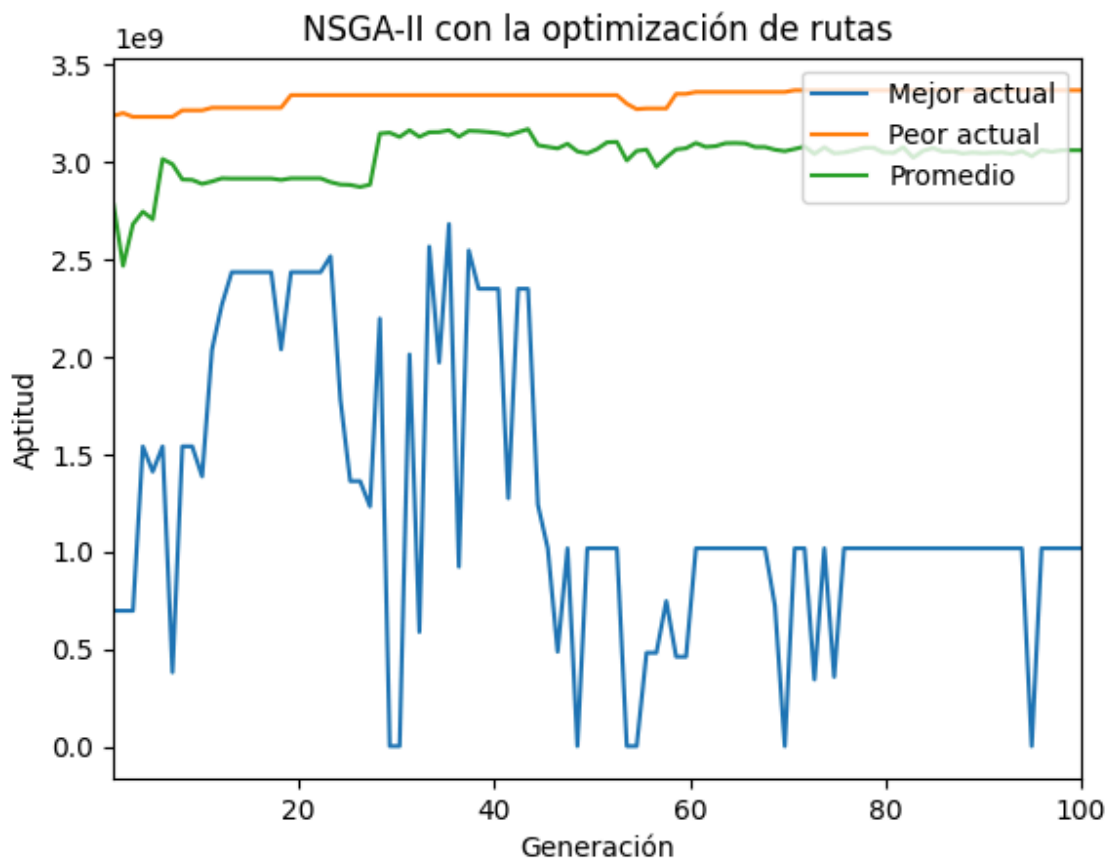


La optimización de rutas en el algoritmo NSGA-II muestra una tendencia interesante en nuestros resultados. Las mejores soluciones convergen rápidamente hacia óptimos locales, manteniendo una calidad consistente a lo largo del proceso. Esto sugiere que el algoritmo es efectivo para explotar soluciones prometedoras.

Por otro lado, aunque el rendimiento en los casos peores y promedio se mantiene relativamente alto, la variabilidad es más pronunciada, indicando que el algoritmo puede tener dificultades para encontrar soluciones óptimas en situaciones más complejas o con datos más grandes. Esto resalta la necesidad de ajustar el algoritmo para mejorar su robustez en escenarios desafiantes.



Sin embargo, podemos analizar en mayor medida el la evolución del mejor valor en la siguiente gráfica, en donde podemos notar que si se buscaron otros valores mejores, pero al final el que se encontró en el algoritmo en su primer cuarto fue la solución más óptima.



En cuanto a la siguiente gráfica que vamos a analizar, se trata de un gráfico tridimensional donde cada coordenada representa la carga, el tiempo y la distancia de una solución. La solución elegida, resaltada en rojo, parece dominar a las demás en términos de tiempo y distancia, ya que son menores que las otras soluciones. Además, podemos observar que la carga de esta solución en particular es una de las más altas, lo cual es un resultado positivo si consideramos que se busca maximizar la carga.

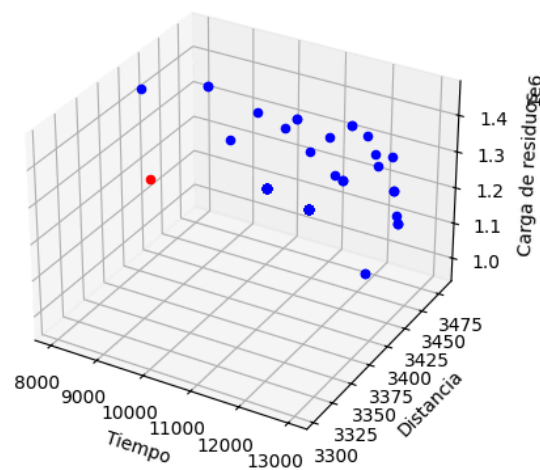
Este hallazgo sugiere que el algoritmo está logrando optimizar otros aspectos, como el tiempo y la distancia, sin comprometer significativamente la capacidad de carga del camión. Es esencial destacar que una carga alta es deseable en este contexto, ya que implica una mayor eficiencia en el transporte de mercancías.

Por otra parte, notamos que en general, las soluciones de la generación después de correr el algoritmo están agrupadas en la esquina del lado de mayor tiempo, distancia y carga. Este agrupamiento puede indicar que el algoritmo ha convergido hacia una

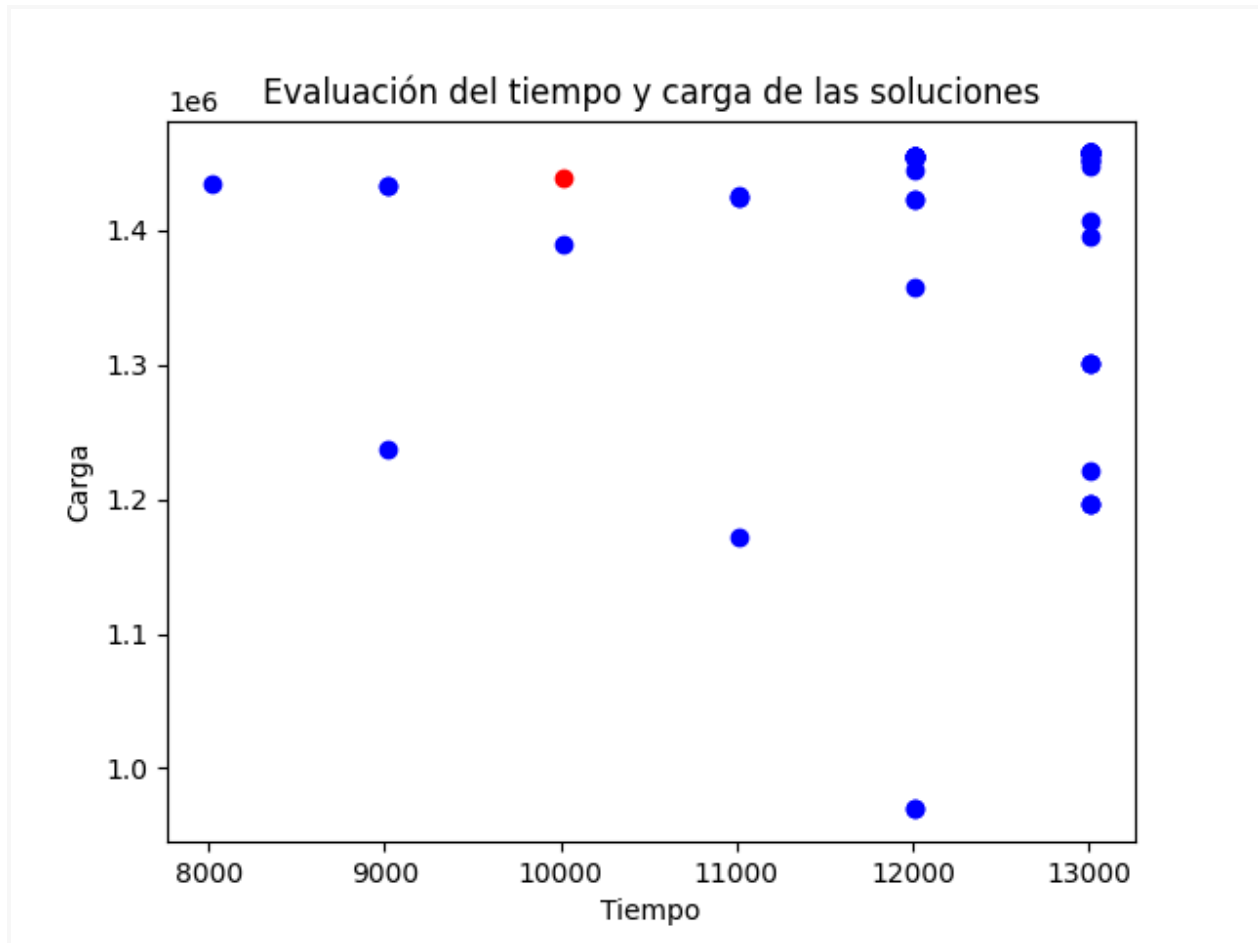
región del espacio de soluciones donde se encuentran soluciones óptimas o cercanas a óptimas en términos de tiempo, distancia y carga.

En resumen, la solución en rojo muestra un equilibrio exitoso entre una carga alta y valores bajos de tiempo y distancia, lo cual es altamente deseable en un problema de optimización de rutas de camiones. Esto sugiere que el algoritmo NSGA-II ha sido efectivo para encontrar soluciones que satisfacen múltiples objetivos de manera simultánea.

Evaluación de todos los objetivos de las soluciones



Observando la solución resaltada en rojo, notamos que esta solución en particular ha obtenido una de las mayores cargas entre todas las soluciones evaluadas en el conjunto. Aunque esta solución no posee los valores más bajos de tiempo, estos valores se sitúan en un rango intermedio en comparación con otras soluciones. Este equilibrio entre una carga alta y valores de tiempo y distancia aceptables sugiere que la solución elegida ha logrado cumplir con los requisitos operativos necesarios para el transporte de residuos de manera eficiente.



Los resultados obtenidos muestran la efectividad del algoritmo NSGA-II en la optimización de rutas de camiones, con el objetivo principal de maximizar la carga transportada. A través del análisis de múltiples soluciones generadas por el algoritmo, se observa que la mayoría de las soluciones alcanzan cargas significativamente altas, lo cual es un indicador positivo ya que maximizar la carga es fundamental para aumentar la eficiencia y rentabilidad del transporte de mercancías.

Además, se evidencia que el algoritmo logra encontrar soluciones equilibradas, donde, si bien algunas soluciones pueden tener tiempos y distancias ligeramente más largos, estas compensaciones se realizan en aras de maximizar la carga transportada. Este equilibrio entre múltiples objetivos demuestra la capacidad del algoritmo para encontrar soluciones eficientes que satisfacen diferentes restricciones operativas.

Se corrió el algoritmo en diferentes ejemplares con una cantidad variable de nodos para una comparación cualitativa del algoritmo, a continuación los resultados de este análisis:

Tabla de resultados de los ejemplares utilizados para el análisis

Ejemplar	Mejor	Peor	Media	Mediana	Desviación estándar
A-n32-k5	1174654.407	1388517.456	1273614.548	1241174.906	75729.383
A-n63-k10	2243692.672	3002220.273	2777170.880	2619370.020	290902.369
E-n101-k14	3154077.742	3175499.851	3164451.108	3169898.184	6982.882
X-n153-k22	1207240343.698	35714475.136	976649752.447	933633519.982	798975727.302
M-n200-k17	6208552.066	6330186.239	6273455.401	6268245.480	48329.863

Conclusiones

El proyecto de implementación del algoritmo multiobjetivo NSGA-II para la optimización de rutas de camiones ha arrojado resultados prometedores y relevantes en términos de eficiencia y rentabilidad en el transporte de mercancías.

Los resultados reflejan el éxito del algoritmo en maximizar la carga transportada, un factor clave en la logística del transporte de mercancías. La mayoría de las soluciones obtenidas alcanzan niveles de carga notables, lo que contribuye significativamente a mejorar la eficiencia y la rentabilidad en este ámbito.

Lo que este proyecto nos muestra es cómo los algoritmos genéticos multiobjetivo pueden encontrar soluciones que equilibran diferentes aspectos, como la maximización de la carga transportada, la minimización de los tiempos de entrega y la reducción de las distancias recorridas.

Es importante mencionar que, en el proceso de optimización de rutas, fue importante para nosotros poder ofrecer una solución en cuando a aspectos éticos y ambientales, como el impacto ambiental de las rutas optimizadas en términos de emisiones de carbono y congestión del tráfico. Estas consideraciones son cruciales para garantizar que las soluciones propuestas sean sostenibles y respetuosas con el medio ambiente.

Podemos decir que el proyecto nos ofreció una perspectiva alentadora sobre el potencial del algoritmo NSGA-II en la planificación de rutas de camiones, proporcionando una herramienta valiosa para mejorar la logística en operaciones de

transporte de residuos y ofreciendo recomendaciones para una implementación práctica y sostenible.

Referencias

- J. Pamungkas and C. W. Wijaya, "Optimization The Waste Management Based on Genetic Algorithm Multiobjective," 2019 IEEE 10th International Conference on Software Engineering and Service Science (ICSESS), Beijing, China, 2019, pp. 1-4, doi: 10.1109/ICSESS47205.2019.9040812.
- Karadimas, N.V., Papatzelou, K., Loumos, V.G. (2007). Genetic Algorithms for Municipal Solid Waste Collection and Routing Optimization. In: Boukis, C., Pnevmatikakis, A., Polymenakos, L. (eds) Artificial Intelligence and Innovations 2007: from Theory to Applications. AIAI 2007. IFIP The International Federation for Information Processing, vol 247. Springer, Boston, MA. https://doi.org/10.1007/978-0-387-74161-1_24
- Alberdi, E.; Urrutia, L.; Goti, A.; Oyarbide-Zubillaga, A. Modeling the Municipal Waste Collection Using Genetic Algorithms. Processes 2020, 8, 513. <https://doi.org/10.3390/pr8050513>
- A. Alwabli and I. Kostanic, "Dynamic Route Optimization For Waste Collection Using Genetic Algorithm," 2020 International Conference on Computer Science and Its Application in Agriculture (ICOSICA), Bogor, Indonesia, 2020, pp. 1-7, doi: 10.1109/ICOSICA49951.2020.9243256.
- K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in IEEE Transactions on Evolutionary Computation, vol. 6, no. 2, pp. 182-197, April 2002, doi: 10.1109/4235.996017.