Tarea opti

In [7]:
```
import random
import numpy as np
import matplotlib.pyplot as plt
import time
```

In [8]:
```
def cargar_datos(nombre_archivo):
    with open(nombre_archivo, 'r') as f:
        lineas = f.readlines()

    n, m = map(int, lineas[0].split())

    f_j = [0] * (n + 1)
    c_ij = [[0] * (n + 1) for _ in range(m + 1)]

    for j in range(1, n + 1):
        datos = list(map(int, lineas[j].split()))
        idx = datos[0]
        f_j[idx] = datos[1]

        for i in range(1, m + 1):
            c_ij[i][idx] = datos[i + 1]

    return n, m, f_j, c_ij

def algoritmo_add(n, m, f_j, c_ij):
    J_estrella = set()
    alloc_i = [0] * (m + 1)

    f = 0

    g = [0] * (n + 1)
    for j in range(1, n + 1):
        g[j] = sum(c_ij[i][j] for i in range(1, m + 1)) + f_j[j]

    j_estrella = 1
    for j in range(2, n + 1):
```

```
        if g[j] < g[j_estrella]:
            j_estrella = j

    J_estrella.add(j_estrella)
    for i in range(1, m + 1):
        alloc_i[i] = j_estrella
    f = g[j_estrella]

    while True:
        g = [0] * (n + 1)
        for j in range(1, n + 1):
            if j not in J_estrella:
                suma = 0
                for i in range(1, m + 1):
                    diferencia = c_ij[i][alloc_i[i]] - c_ij[i][j]
                    if diferencia > 0:
                        suma += diferencia
                g[j] = suma - f_j[j]

        mejor_j = -1
        mejor_g = float('-inf')
        for j in range(1, n + 1):
            if j not in J_estrella and g[j] > mejor_g:
                mejor_j = j
                mejor_g = g[j]

        if mejor_j == -1 or mejor_g <= 0:
            break

        j_estrella = mejor_j
        J_estrella.add(j_estrella)

        for i in range(1, m + 1):
            if c_ij[i][j_estrella] < c_ij[i][alloc_i[i]]:
                alloc_i[i] = j_estrella

        f = f - mejor_g

    costo_fijo = sum(f_j[j] for j in J_estrella)
    costo_asignacion = sum(c_ij[i][alloc_i[i]] for i in range(1, m + 1))
    costo_total = costo_fijo + costo_asignacion
```

```
    return J_estrella, alloc_i, costo_total

def algoritmo_semivoraz(n, m, f_j, c_ij, alpha):
    J_estrella = set()

    alloc_i = [0] * (m + 1)

    f = 0


    g = [0] * (n + 1)
    for j in range(1, n + 1):
        g[j] = sum(c_ij[i][j] for i in range(1, m + 1)) + f_j[j]


    mejor_j = 1
    peor_j = 1
    for j in range(2, n + 1):
        if g[j] < g[mejor_j]:
            mejor_j = j
        if g[j] > g[peor_j]:
            peor_j = j

    g_min = g[mejor_j]
    g_max = g[peor_j]


    umbral = g_min + alpha * (g_max - g_min)
    lista_candidatos = [j for j in range(1, n + 1) if g[j] <= umbral]


    j_estrella = random.choice(lista_candidatos)


    J_estrella.add(j_estrella)
    for i in range(1, m + 1):
        alloc_i[i] = j_estrella
    f = g[j_estrella]


    while True:
        g = [0] * (n + 1)
```

```
        for j in range(1, n + 1):
            if j not in J_estrella:
                suma = 0
                for i in range(1, m + 1):
                    diferencia = c_ij[i][alloc_i[i]] - c_ij[i][j]
                    if diferencia > 0:
                        suma += diferencia
                g[j] = suma - f_j[j]


        candidatos = [j for j in range(1, n + 1) if j not in J_estrella and g[j] > 0]
        if not candidatos:
            break


        g_candidatos = [g[j] for j in candidatos]
        if not g_candidatos:
            break

        g_min = min(g_candidatos)
        g_max = max(g_candidatos)


        if g_min == g_max:
            lista_candidatos = candidatos
        else:
            umbral = g_max - alpha * (g_max - g_min)
            lista_candidatos = [j for j in candidatos if g[j] >= umbral]


        if lista_candidatos:
            j_estrella = random.choice(lista_candidatos)
            J_estrella.add(j_estrella)


            for i in range(1, m + 1):
                if c_ij[i][j_estrella] < c_ij[i][alloc_i[i]]:
                    alloc_i[i] = j_estrella

            f = f - g[j_estrella]
        else:
            break
```

```
    costo_fijo = sum(f_j[j] for j in J_estrella)
    costo_asignacion = sum(c_ij[i][alloc_i[i]] for i in range(1, m + 1))
    costo_total = costo_fijo + costo_asignacion

    return J_estrella, alloc_i, costo_total


def busqueda_local(J_estrella, alloc_i, n, m, f_j, c_ij):

    costo_fijo = sum(f_j[j] for j in J_estrella)
    costo_asignacion = sum(c_ij[i][alloc_i[i]] for i in range(1, m + 1))
    costo_actual = costo_fijo + costo_asignacion


    J_estrella = list(J_estrella)
    mejora = True
    max_iteraciones = 20
    iteracion = 0

    while mejora and iteracion < max_iteraciones:
        iteracion += 1
        mejora = False

        random.shuffle(J_estrella)
        j_quitar_list = J_estrella[:min(5, len(J_estrella))]

        for j_quitar in j_quitar_list:
            if len(J_estrella) <= 1:
                break

            j_añadir_candidates = [j for j in range(1, n + 1) if j not in J_estrella]
            random.shuffle(j_añadir_candidates)
            j_añadir_list = j_añadir_candidates[:min(10, len(j_añadir_candidates))]

            for j_añadir in j_añadir_list:
                temp_J_estrella = J_estrella.copy()
                temp_J_estrella.remove(j_quitar)
                temp_J_estrella.append(j_añadir)
                temp_alloc_i = alloc_i.copy()
```

```
                for i in range(1, m + 1):
                    if temp_alloc_i[i] == j_quitar:
                        mejor_j = temp_J_estrella[0]
                        mejor_costo = c_ij[i][mejor_j]

                        for j in temp_J_estrella[1:]:
                            if c_ij[i][j] < mejor_costo:
                                mejor_costo = c_ij[i][j]
                                mejor_j = j

                        temp_alloc_i[i] = mejor_j
                    elif c_ij[i][j_añadir] < c_ij[i][temp_alloc_i[i]]:
                        temp_alloc_i[i] = j_añadir

                nuevo_costo_fijo = sum(f_j[j] for j in temp_J_estrella)
                nuevo_costo_asignacion = sum(c_ij[i][temp_alloc_i[i]] for i in range(1, m + 1))
                nuevo_costo = nuevo_costo_fijo + nuevo_costo_asignacion

                if nuevo_costo < costo_actual:
                    J_estrella = temp_J_estrella
                    alloc_i = temp_alloc_i
                    costo_actual = nuevo_costo
                    mejora = True
                    break

            if mejora:
                break

    return set(J_estrella), alloc_i, costo_actual


def algoritmo_semivoraz_multiarranque(n, m, f_j, c_ij, alpha, num_iteraciones=200):

    mejor_costo = float('inf')
    mejor_J_estrella = None
    mejor_alloc_i = None



    resultados = []
```

```
    for _ in range(num_iteraciones):

        J_estrella, alloc_i, costo_total = algoritmo_semivoraz(n, m, f_j, c_ij, alpha)


        resultados.append(costo_total)


        if costo_total < mejor_costo:
            mejor_costo = costo_total
            mejor_J_estrella = J_estrella.copy()
            mejor_alloc_i = alloc_i.copy()

    return mejor_J_estrella, mejor_alloc_i, mejor_costo, resultados


def algoritmo_grasp(n, m, f_j, c_ij, alpha, max_iter_sin_mejora=50):

    mejor_costo_global = float('inf')
    mejor_J_estrella = None
    mejor_alloc_i = None


    iter_sin_mejora = 0
    total_iteraciones = 0


    resultados = []


    while iter_sin_mejora < max_iter_sin_mejora:
        total_iteraciones += 1


        J_estrella, alloc_i, costo_constructivo = algoritmo_semivoraz(n, m, f_j, c_ij, alpha)


        J_estrella_mejorado, alloc_i_mejorado, costo_mejorado = busqueda_local(
            J_estrella, alloc_i, n, m, f_j, c_ij)


        resultados.append(costo_mejorado)
```

```
        if costo_mejorado < mejor_costo_global:
            mejor_costo_global = costo_mejorado
            mejor_J_estrella = J_estrella_mejorado.copy()
            mejor_alloc_i = alloc_i_mejorado.copy()
            iter_sin_mejora = 0
        else:
            iter_sin_mejora += 1

    return mejor_J_estrella, mejor_alloc_i, mejor_costo_global, resultados, total_iteraciones


def crear_histogramas(resultados_por_alpha, valores_alpha, nombre_algoritmo, num_iteraciones):
    plt.figure(figsize=(12, 8))

    colores = ['blue', 'green', 'red']
    etiquetas = [f'α = {alpha} ({num_iteraciones[i]} iter.)' for i, alpha in enumerate(valores_alpha)]


    todos_valores = []
    for resultados in resultados_por_alpha:
        todos_valores.extend(resultados)

    min_valor = min(todos_valores)
    max_valor = max(todos_valores)


    bin_range = max_valor - min_valor
    num_bins = 20
    bin_size = bin_range / num_bins if bin_range > 0 else 1
    bins = [min_valor + i * bin_size for i in range(num_bins + 1)]


    for i, resultados in enumerate(resultados_por_alpha):
        plt.hist(resultados, bins=bins, alpha=0.7,
                 color=colores[i], label=etiquetas[i], edgecolor='black')

    plt.title(f'Distribución de resultados {nombre_algoritmo} para diferentes valores de α', fontsize=14)
    plt.xlabel('Valor de la función objetivo (Costo total)', fontsize=12)
    plt.ylabel('Frecuencia', fontsize=12)
    plt.legend(fontsize=12)
```

```
    plt.grid(True, linestyle='--', alpha=0.7)


    for i, resultados in enumerate(resultados_por_alpha):
        mejor_valor = min(resultados)
        plt.axvline(x=mejor_valor, color=colores[i], linestyle='--',
                    label=f'Mejor valor para α={valores_alpha[i]}: {mejor_valor}')

    plt.tight_layout()
    plt.savefig(f'histograma_{nombre_algoritmo}.png')
    plt.show()
```

In [9]:

```
def ejecutar(nombre_archivo, optimo):

    n, m, f_j, c_ij = cargar_datos(nombre_archivo)

    print(f"Problema con {n} ubicaciones potenciales y {m} puntos de demanda")


    alphas = [0.05, 0.50, 0.95]


    print("\n" + "="*80)
    print("PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE")
    print("="*80)

    resultados_multiarranque = []
    mejores_soluciones_multiarranque = []
    num_iter_multiarranque = []

    for alpha in alphas:
        print(f"\nEjecutando algoritmo semi-voraz multi-arranque con alpha = {alpha}:")


        inicio = time.time()
        mejor_J_estrella, mejor_alloc_i, mejor_costo, resultados = algoritmo_semivoraz_multiarranque(
            n, m, f_j, c_ij, alpha)
        fin = time.time()

        resultados_multiarranque.append(resultados)
        mejores_soluciones_multiarranque.append((mejor_J_estrella, mejor_alloc_i, mejor_costo))
        num_iter_multiarranque.append(len(resultados))
```

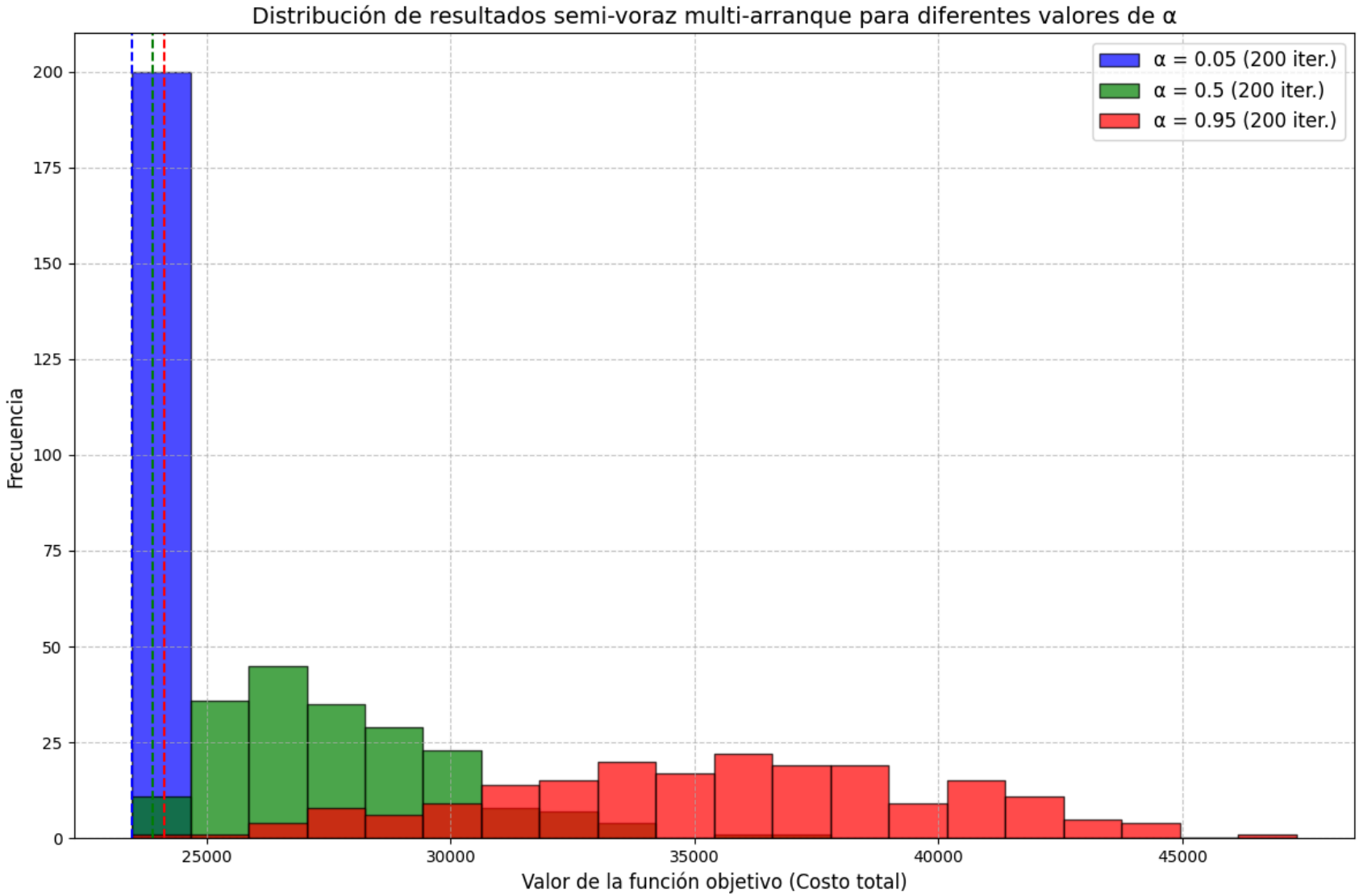
```
    print(f"  Mejor solución encontrada:")
    print(f"  Instalaciones seleccionadas: {sorted(mejor_J_estrella)}")
    print(f"  Costo total: {mejor_costo}")
    print(f"  Número de iteraciones: {len(resultados)}")
    print(f"  Tiempo de ejecución: {fin - inicio:.2f} segundos")


    print(f"  Estadísticas de las {len(resultados)} iteraciones:")
    print(f"    Mejor: {min(resultados)}")
    print(f"    Peor: {max(resultados)}")
    print(f"    Promedio: {sum(resultados)/len(resultados):.2f}")
    print(f"    Desviación porcentual: {((min(resultados)-optimo)/optimo)*100:.2f}%")


crear_histogramas(resultados_multiarranque, alphas, "semi-voraz multi-arranque", num_iter_multiarranque)


print("\n" + "="*80)
print("PARTE 2: ALGORITMO GRASP")
print("="*80)

resultados_grasp = []
mejores_soluciones_grasp = []
num_iter_grasp = []

for alpha in alphas:
    print(f"\nEjecutando algoritmo GRASP con alpha = {alpha}:")


    inicio = time.time()
    mejor_J_estrella, mejor_alloc_i, mejor_costo, resultados, total_iter = algoritmo_grasp(
        n, m, f_j, c_ij, alpha)
    fin = time.time()

    resultados_grasp.append(resultados)
    mejores_soluciones_grasp.append((mejor_J_estrella, mejor_alloc_i, mejor_costo))
    num_iter_grasp.append(total_iter)


    print(f"  Mejor solución encontrada:")
```

```
        print(f"  Instalaciones seleccionadas: {sorted(mejor_J_estrella)}")
        print(f"  Costo total: {mejor_costo}")
        print(f"  Total de iteraciones: {total_iter}")
        print(f"  Tiempo de ejecución: {fin - inicio:.2f} segundos")


        print(f"  Estadísticas de las {len(resultados)} iteraciones:")
        print(f"    Mejor: {min(resultados)}")
        print(f"    Peor: {max(resultados)}")
        print(f"    Promedio: {sum(resultados)/len(resultados):.2f}")
        print(f"    Desviación porcentual: {((min(resultados)-optimo)/optimo)*100:.2f}%")


    crear_histogramas(resultados_grasp, alphas, "GRASP", num_iter_grasp)


    print("\n" + "="*80)
    print("MEJOR SOLUCIÓN GLOBAL")
    print("="*80)


    mejor_costo_multiarranque = min([solucion[2] for solucion in mejores_soluciones_multiarranque])
    mejor_costo_grasp = min([solucion[2] for solucion in mejores_soluciones_grasp])

    if mejor_costo_grasp <= mejor_costo_multiarranque:
        mejor_algo = "GRASP"
        mejor_idx = [solucion[2] for solucion in mejores_soluciones_grasp].index(mejor_costo_grasp)
        mejor_alpha = alphas[mejor_idx]
        mejor_solucion = mejores_soluciones_grasp[mejor_idx]
    else:
        mejor_algo = "Semi-voraz multi-arranque"
        mejor_idx = [solucion[2] for solucion in mejores_soluciones_multiarranque].index(mejor_costo_multiarranque)
        mejor_alpha = alphas[mejor_idx]
        mejor_solucion = mejores_soluciones_multiarranque[mejor_idx]

    print(f"La mejor solución global fue encontrada por el algoritmo {mejor_algo} con alpha = {mejor_alpha}:")
    print(f"  Instalaciones seleccionadas: {sorted(mejor_solucion[0])}")
    print(f"  Costo total: {mejor_solucion[2]}")
```

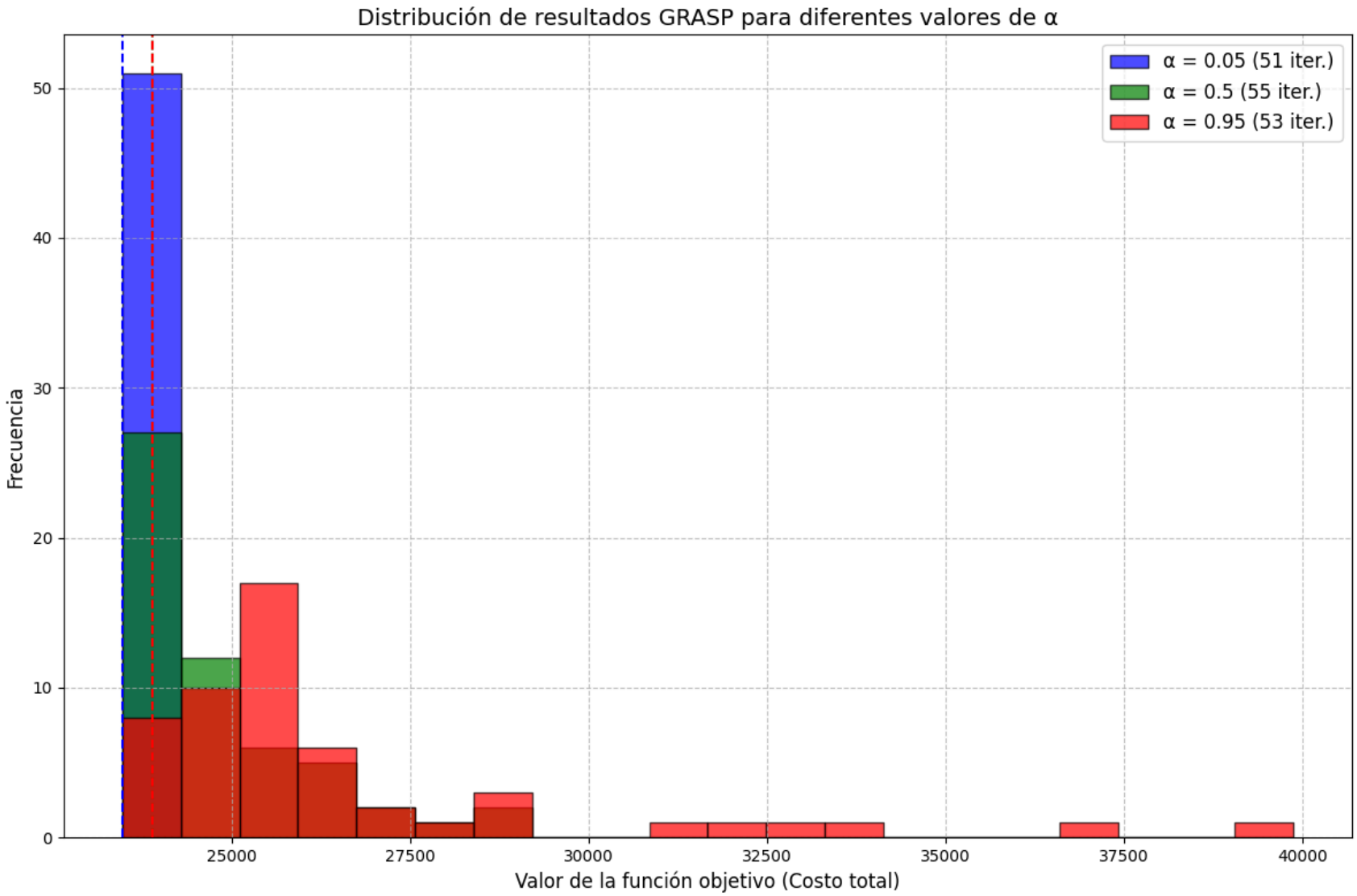
Instancia 1

In [10]:
```
ejecutar("UFLP-1.txt", 23468)
```

```
Problema con 50 ubicaciones potenciales y 100 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [11, 14, 15, 33, 44]
  Costo total: 23468
  Número de iteraciones: 200
  Tiempo de ejecución: 0.52 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 23468
    Peor: 23468
    Promedio: 23468.00
    Desviación porcentual: 0.00%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [11, 14, 15, 26, 33, 44]
  Costo total: 23897
  Número de iteraciones: 200
  Tiempo de ejecución: 0.63 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 23897
    Peor: 36631
    Promedio: 27682.49
    Desviación porcentual: 1.83%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [11, 14, 15, 26, 32, 44]
  Costo total: 24128
  Número de iteraciones: 200
  Tiempo de ejecución: 0.68 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 24128
    Peor: 47337
    Promedio: 35525.82
    Desviación porcentual: 2.81%
```


Distribución de resultados semi-voraz multi-arranque para diferentes valores de α
α = 0.05 (200 iter.)
α = 0.5 (200 iter.)
α = 0.95 (200 iter.)
Frecuencia
0
25
50
75
100
125
150
175
200
25000
30000
35000
40000
45000
Valor de la función objetivo (Costo total)

```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [11, 14, 15, 33, 44]
  Costo total: 23468
  Total de iteraciones: 51
  Tiempo de ejecución: 0.20 segundos
  Estadísticas de las 51 iteraciones:
    Mejor: 23468
    Peor: 23468
    Promedio: 23468.00
    Desviación porcentual: 0.00%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [11, 14, 15, 26, 33, 44]
  Costo total: 23897
  Total de iteraciones: 55
  Tiempo de ejecución: 0.33 segundos
  Estadísticas de las 55 iteraciones:
    Mejor: 23897
    Peor: 28998
    Promedio: 24898.71
    Desviación porcentual: 1.83%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [11, 14, 15, 26, 33, 44]
  Costo total: 23897
  Total de iteraciones: 53
  Tiempo de ejecución: 0.39 segundos
  Estadísticas de las 53 iteraciones:
    Mejor: 23897
    Peor: 39873
    Promedio: 26571.40
    Desviación porcentual: 1.83%
```

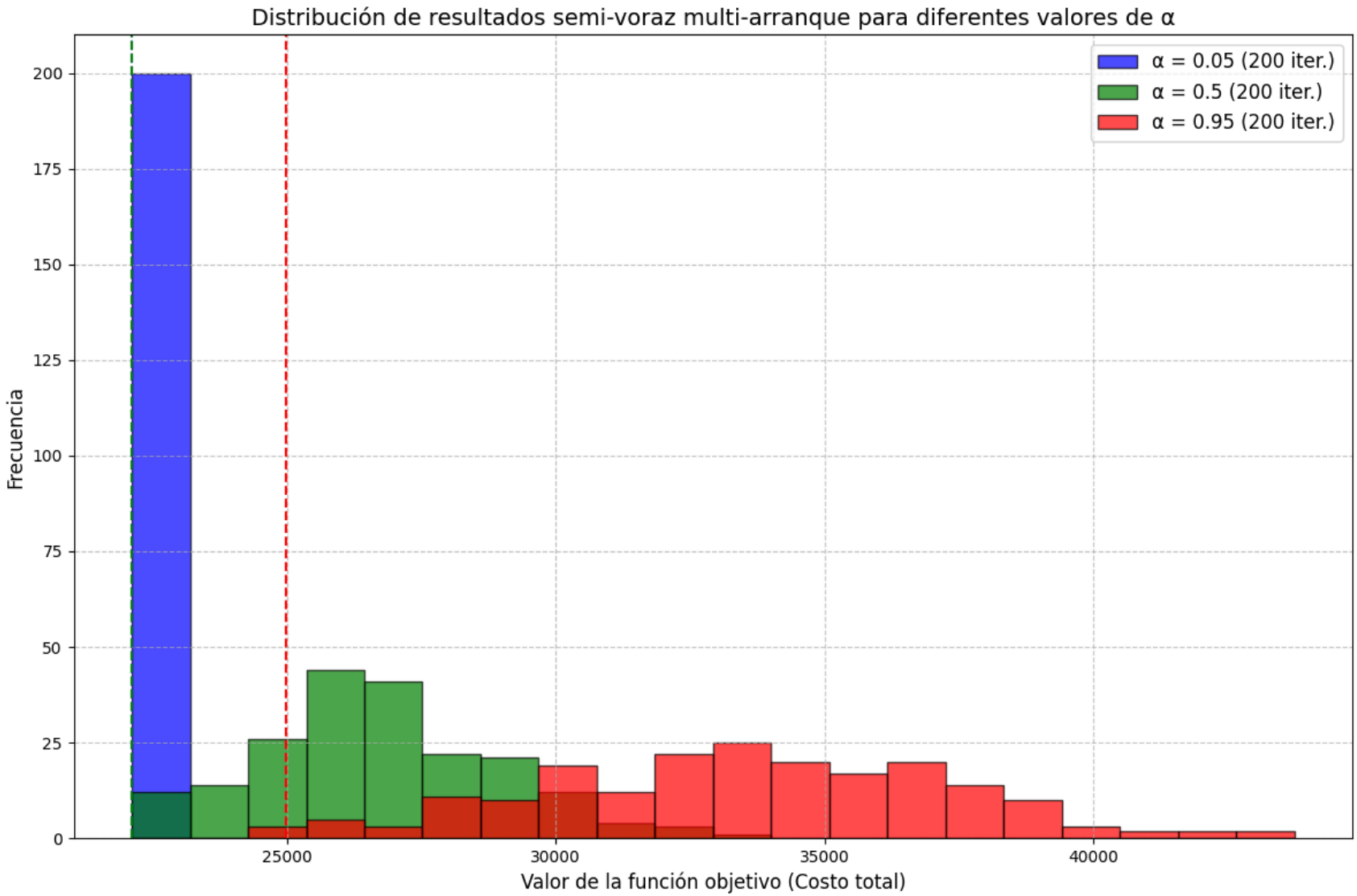



```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo GRASP con alpha = 0.05:
  Instalaciones seleccionadas: [11, 14, 15, 33, 44]
  Costo total: 23468
```

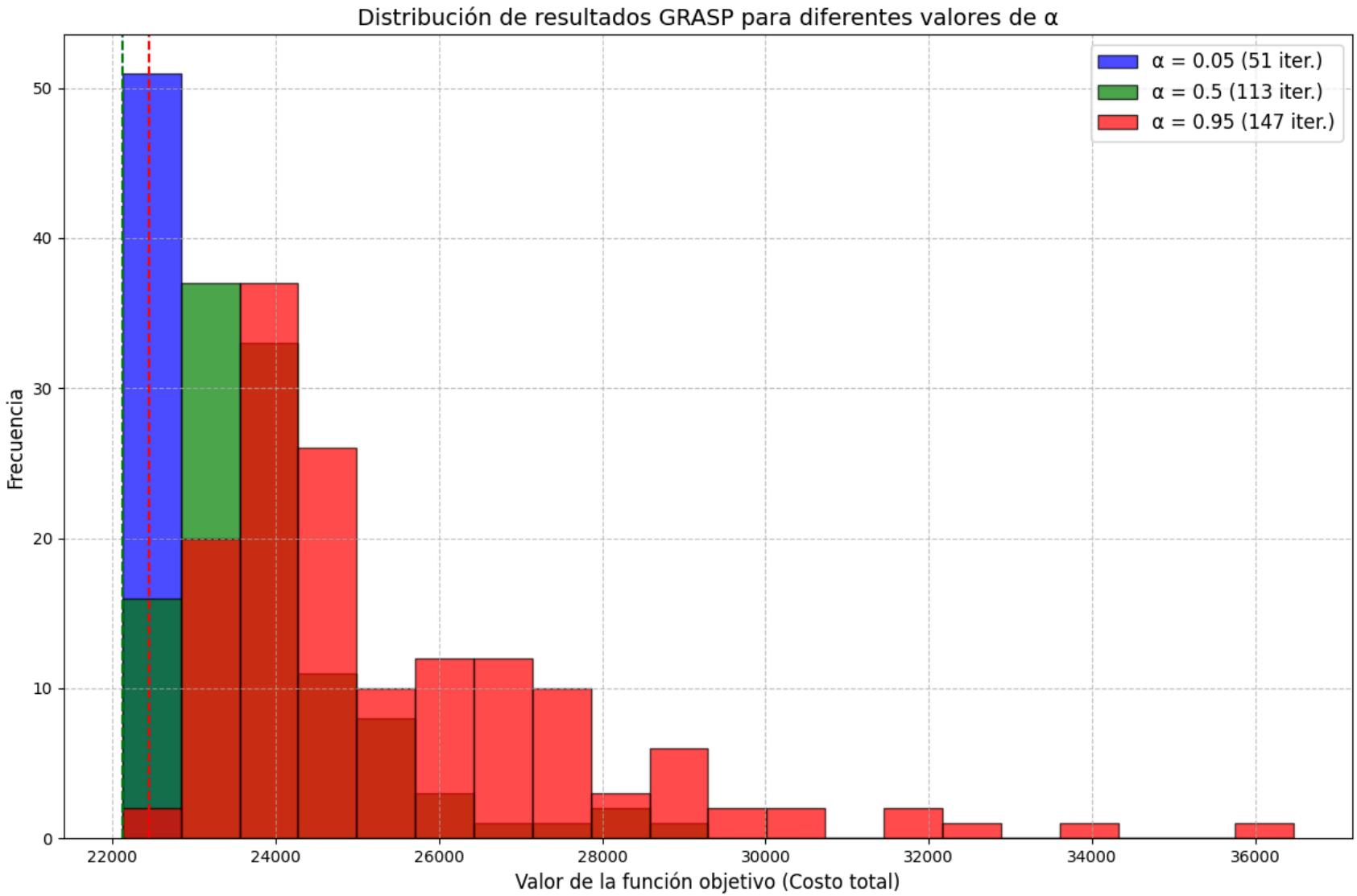
Instancia 2

In [11]:
```
ejecutar("UFLP-2.txt", 22119)
```

```
Problema con 50 ubicaciones potenciales y 100 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [5, 17, 24, 28, 31]
  Costo total: 22119
  Número de iteraciones: 200
  Tiempo de ejecución: 0.51 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 22119
    Peor: 22364
    Promedio: 22241.50
    Desviación porcentual: 0.00%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [5, 17, 24, 28, 31]
  Costo total: 22119
  Número de iteraciones: 200
  Tiempo de ejecución: 0.63 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 22119
    Peor: 33214
    Promedio: 26714.88
    Desviación porcentual: 0.00%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [6, 7, 13, 17, 24, 25, 30]
  Costo total: 24981
  Número de iteraciones: 200
  Tiempo de ejecución: 0.70 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 24981
    Peor: 43739
    Promedio: 33552.79
    Desviación porcentual: 12.94%
```


Distribución de resultados semi-voraz multi-arranque para diferentes valores de α
α = 0.05 (200 iter.)
α = 0.5 (200 iter.)
α = 0.95 (200 iter.)
Frecuencia
0
25
50
75
100
125
150
175
200
25000
30000
35000
40000
Valor de la función objetivo (Costo total)

```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [5, 17, 24, 28, 31]
  Costo total: 22119
  Total de iteraciones: 51
  Tiempo de ejecución: 0.21 segundos
  Estadísticas de las 51 iteraciones:
    Mejor: 22119
    Peor: 22364
    Promedio: 22205.47
    Desviación porcentual: 0.00%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [5, 17, 24, 28, 31]
  Costo total: 22119
  Total de iteraciones: 113
  Tiempo de ejecución: 0.70 segundos
  Estadísticas de las 113 iteraciones:
    Mejor: 22119
    Peor: 29252
    Promedio: 23825.91
    Desviación porcentual: 0.00%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [5, 17, 24, 25, 28, 31]
  Costo total: 22440
  Total de iteraciones: 147
  Tiempo de ejecución: 1.10 segundos
  Estadísticas de las 147 iteraciones:
    Mejor: 22440
    Peor: 36472
    Promedio: 25460.39
    Desviación porcentual: 1.45%
```

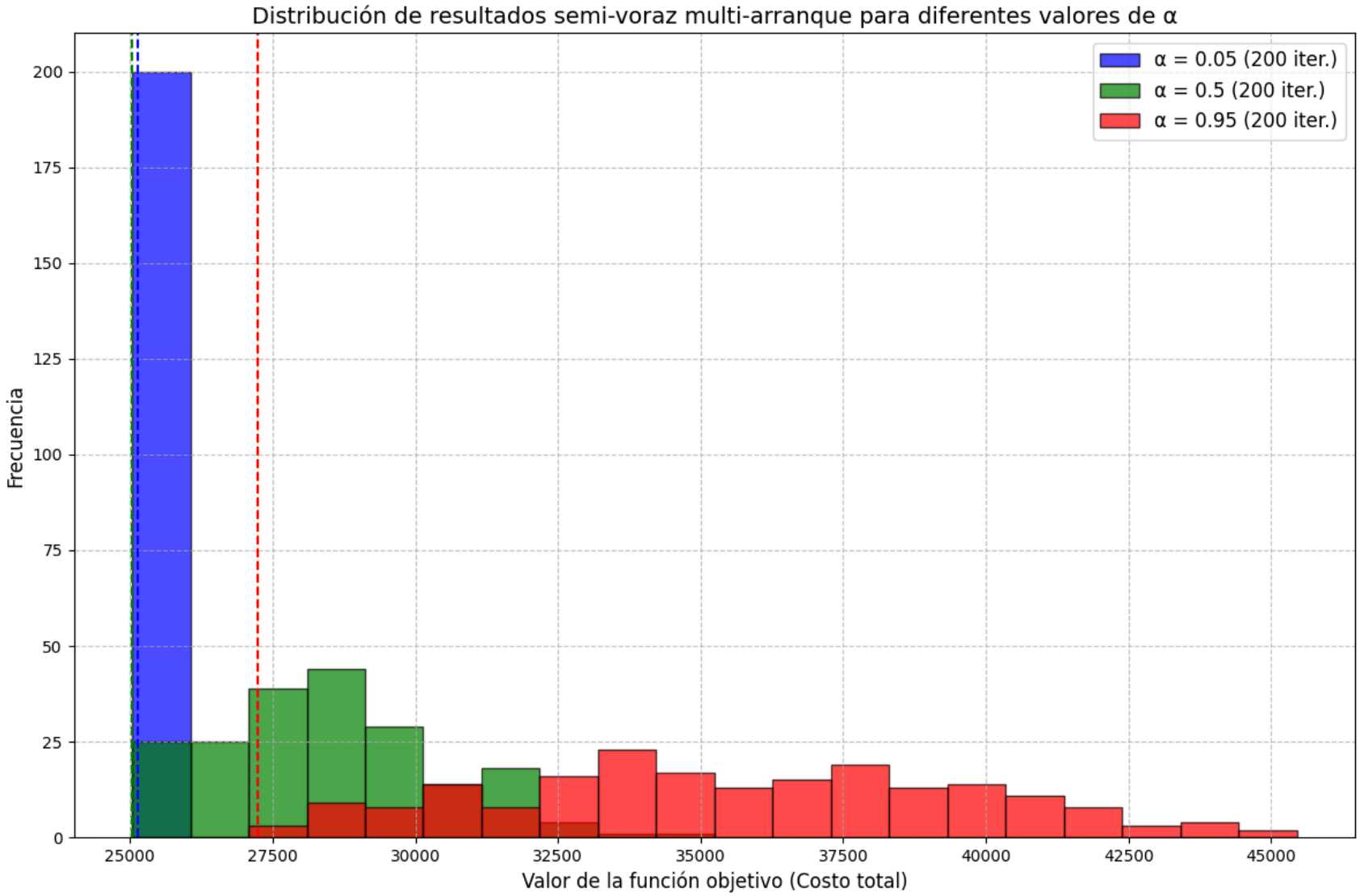



```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo GRASP con alpha = 0.05:
  Instalaciones seleccionadas: [5, 17, 24, 28, 31]
  Costo total: 22119
```

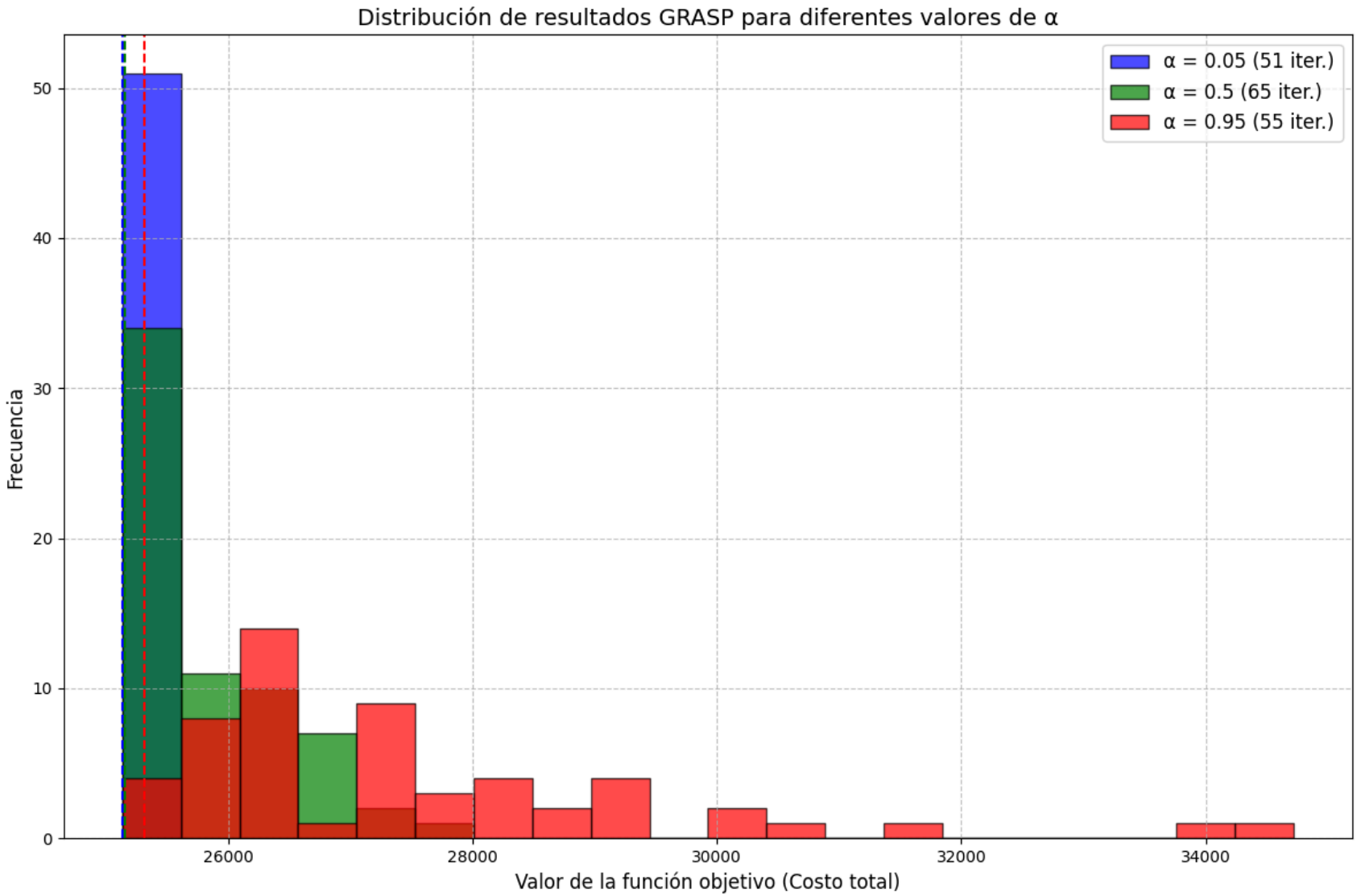
Instancia 3

In [12]:
```
ejecutar("UFLP-3.txt", 25038)
```

```
Problema con 50 ubicaciones potenciales y 100 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [18, 33, 42, 47]
  Costo total: 25132
  Número de iteraciones: 200
  Tiempo de ejecución: 0.44 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 25132
    Peor: 25132
    Promedio: 25132.00
    Desviación porcentual: 0.38%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [10, 33, 42, 48]
  Costo total: 25038
  Número de iteraciones: 200
  Tiempo de ejecución: 0.56 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 25038
    Peor: 35166
    Promedio: 28460.83
    Desviación porcentual: 0.00%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [22, 33, 42, 47, 49]
  Costo total: 27233
  Número de iteraciones: 200
  Tiempo de ejecución: 0.63 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 27233
    Peor: 45449
    Promedio: 35568.66
    Desviación porcentual: 8.77%
```


Distribución de resultados semi-voraz multi-arranque para diferentes valores de α
α = 0.05 (200 iter.)
α = 0.5 (200 iter.)
α = 0.95 (200 iter.)
Frecuencia
0
25
50
75
100
125
150
175
200
25000
27500
30000
32500
35000
37500
40000
42500
45000
Valor de la función objetivo (Costo total)

```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [18, 33, 42, 47]
  Costo total: 25132
  Total de iteraciones: 51
  Tiempo de ejecución: 0.17 segundos
  Estadísticas de las 51 iteraciones:
    Mejor: 25132
    Peor: 25132
    Promedio: 25132.00
    Desviación porcentual: 0.38%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [18, 33, 42, 47, 48]
  Costo total: 25154
  Total de iteraciones: 65
  Tiempo de ejecución: 0.38 segundos
  Estadísticas de las 65 iteraciones:
    Mejor: 25154
    Peor: 27698
    Promedio: 25822.57
    Desviación porcentual: 0.46%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [2, 33, 42, 47, 49]
  Costo total: 25312
  Total de iteraciones: 55
  Tiempo de ejecución: 0.41 segundos
  Estadísticas de las 55 iteraciones:
    Mejor: 25312
    Peor: 34721
    Promedio: 27479.53
    Desviación porcentual: 1.09%
```

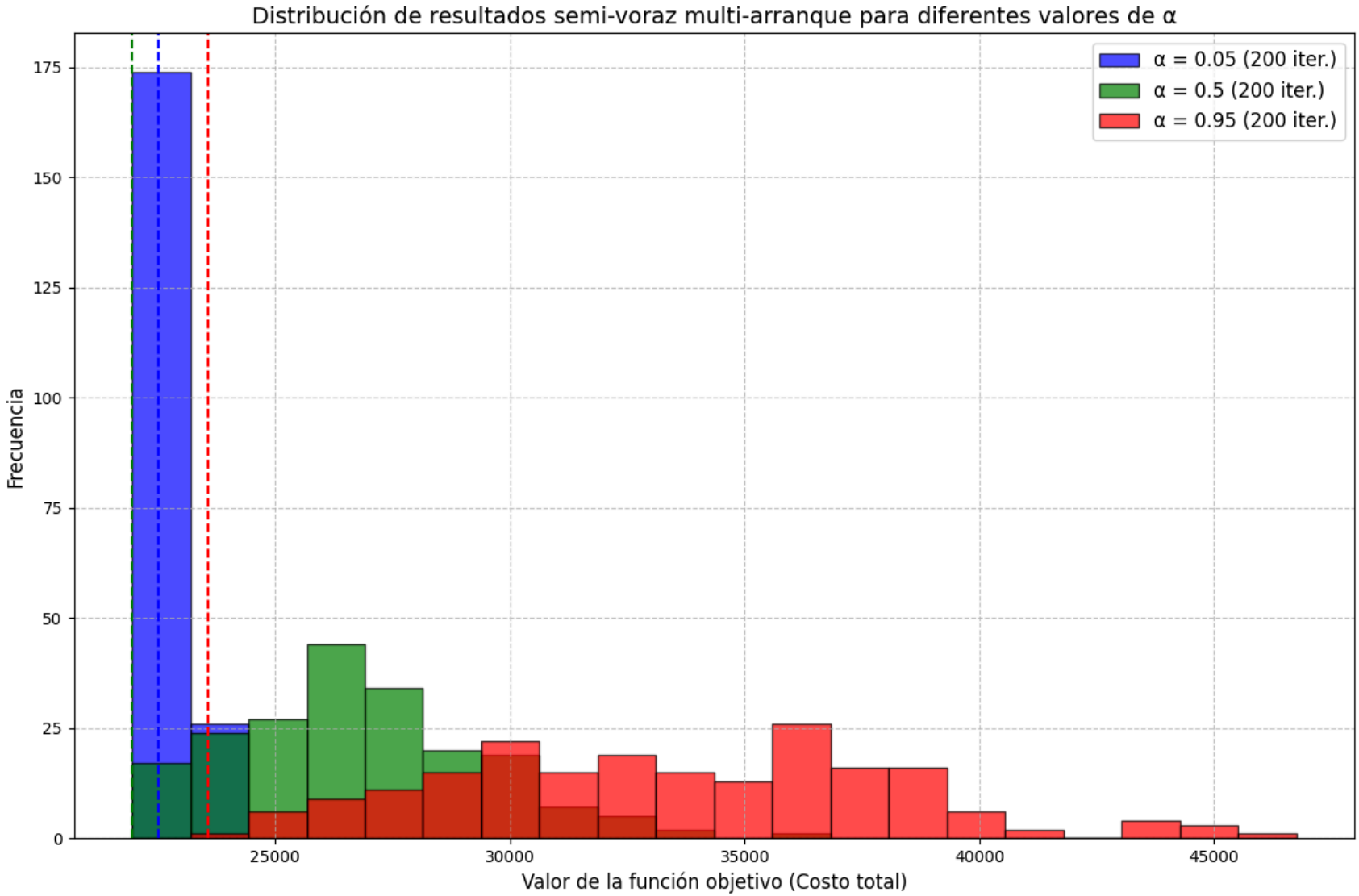



```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo Semi-voraz multi-arranque con alpha = 0.5:
  Instalaciones seleccionadas: [10, 33, 42, 48]
  Costo total: 25038
```

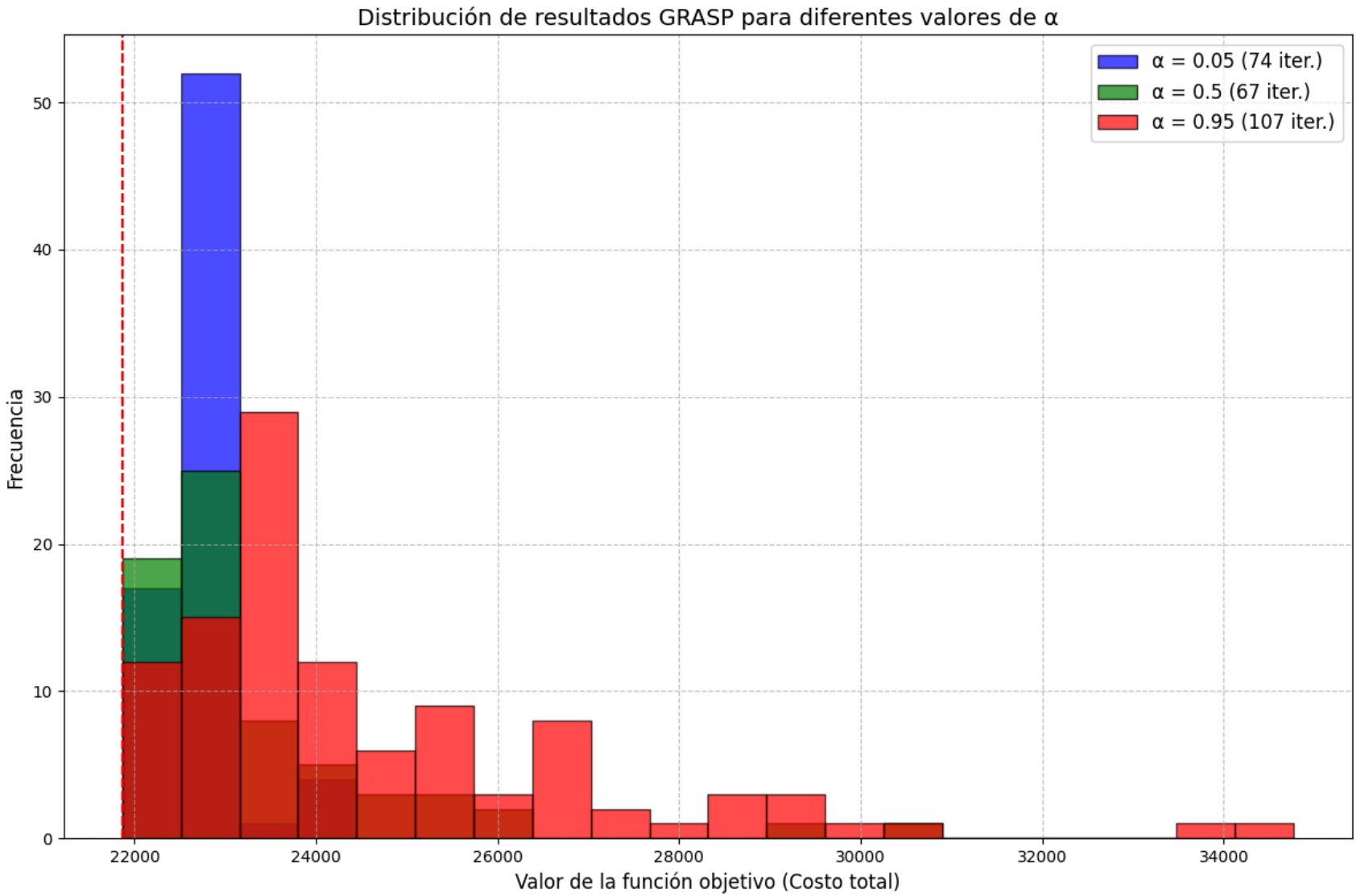
Instancia 4

In [13]:
```
ejecutar("UFLP-4.txt", 21864)
```

```
Problema con 50 ubicaciones potenciales y 100 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [1, 2, 4, 10, 45, 47]
  Costo total: 22515
  Número de iteraciones: 200
  Tiempo de ejecución: 0.63 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 22515
    Peor: 24145
    Promedio: 22746.40
    Desviación porcentual: 2.98%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [1, 2, 4, 10, 22, 47]
  Costo total: 21952
  Número de iteraciones: 200
  Tiempo de ejecución: 0.71 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 21952
    Peor: 35696
    Promedio: 26746.79
    Desviación porcentual: 0.40%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [2, 4, 10, 15, 27, 45, 47]
  Costo total: 23577
  Número de iteraciones: 200
  Tiempo de ejecución: 0.81 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 23577
    Peor: 46745
    Promedio: 33405.79
    Desviación porcentual: 7.83%
```


Distribución de resultados semi-voraz multi-arranque para diferentes valores de α
α = 0.05 (200 iter.)
α = 0.5 (200 iter.)
α = 0.95 (200 iter.)
Frecuencia
0
25
50
75
100
125
150
175
25000
30000
35000
40000
45000
Valor de la función objetivo (Costo total)

```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [1, 4, 10, 22, 47, 49]
  Costo total: 21864
  Total de iteraciones: 74
  Tiempo de ejecución: 0.37 segundos
  Estadísticas de las 74 iteraciones:
    Mejor: 21864
    Peor: 24145
    Promedio: 22496.73
    Desviación porcentual: 0.00%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [1, 4, 10, 22, 47, 49]
  Costo total: 21864
  Total de iteraciones: 67
  Tiempo de ejecución: 0.47 segundos
  Estadísticas de las 67 iteraciones:
    Mejor: 21864
    Peor: 30865
    Promedio: 23292.93
    Desviación porcentual: 0.00%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [1, 4, 10, 22, 47, 49]
  Costo total: 21864
  Total de iteraciones: 107
  Tiempo de ejecución: 0.95 segundos
  Estadísticas de las 107 iteraciones:
    Mejor: 21864
    Peor: 34771
    Promedio: 24622.01
    Desviación porcentual: 0.00%
```

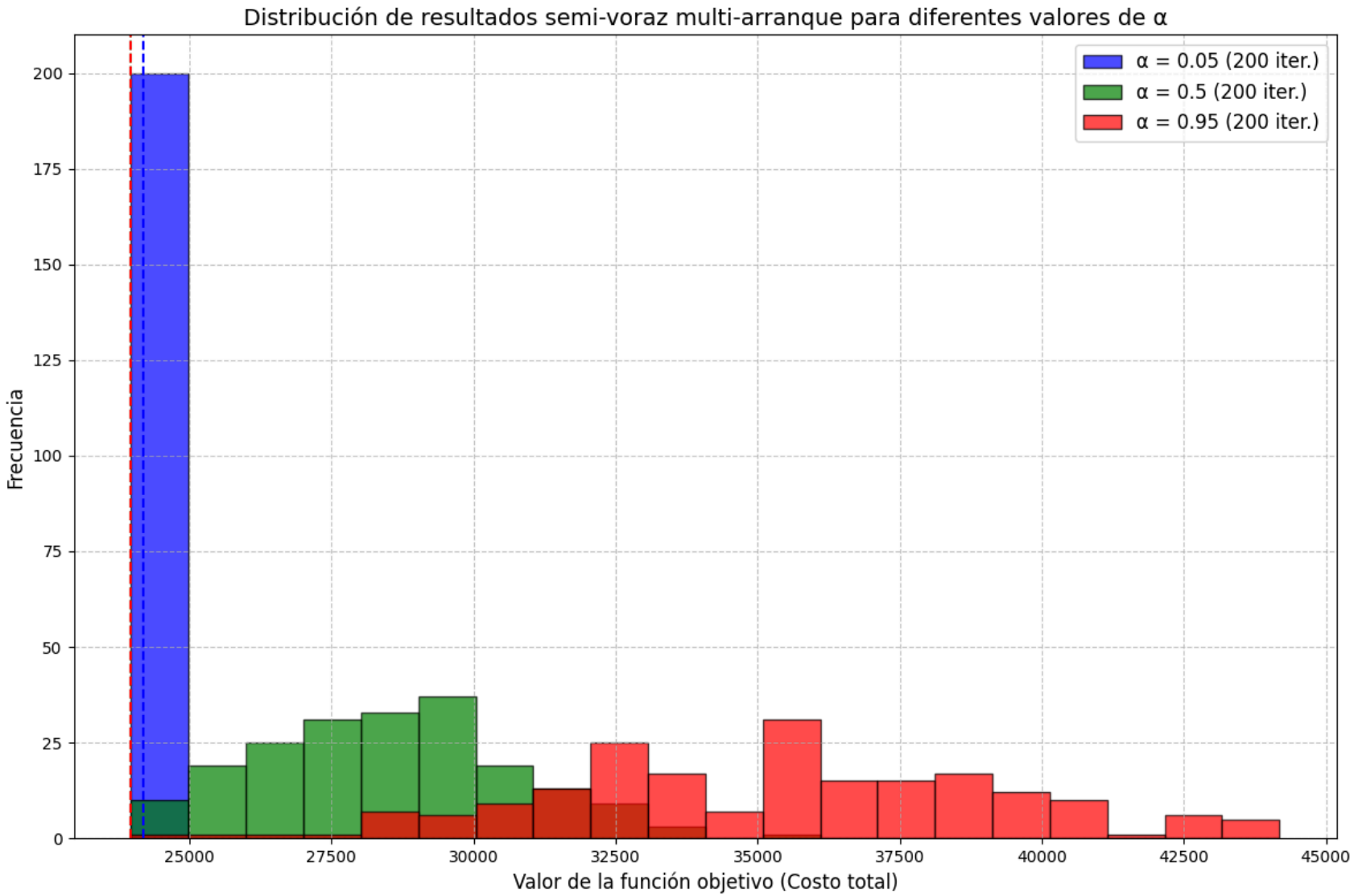



```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo GRASP con alpha = 0.05:
  Instalaciones seleccionadas: [1, 4, 10, 22, 47, 49]
  Costo total: 21864
```

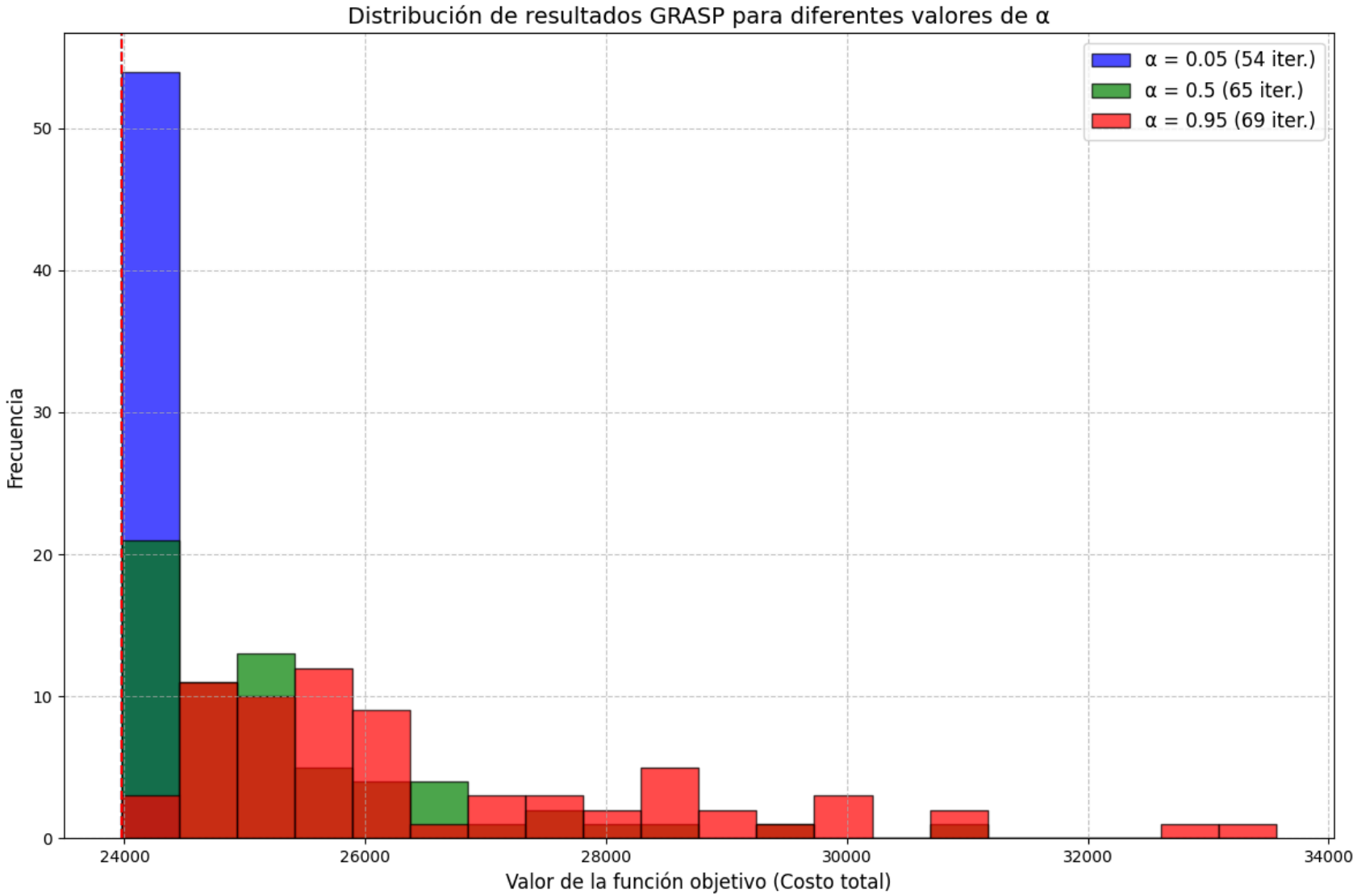
Instancia 5

In [14]:
```
ejecutar("UFLP-5.txt", 23976)
```

```
Problema con 50 ubicaciones potenciales y 100 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 20, 32, 40, 47]
  Costo total: 24198
  Número de iteraciones: 200
  Tiempo de ejecución: 0.64 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 24198
    Peor: 24198
    Promedio: 24198.00
    Desviación porcentual: 0.93%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 20, 26, 32, 47]
  Costo total: 23976
  Número de iteraciones: 200
  Tiempo de ejecución: 0.69 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 23976
    Peor: 35359
    Promedio: 28468.57
    Desviación porcentual: 0.00%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 20, 26, 32, 47]
  Costo total: 23976
  Número de iteraciones: 200
  Tiempo de ejecución: 0.76 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 23976
    Peor: 44182
    Promedio: 35327.92
    Desviación porcentual: 0.00%
```


Distribución de resultados semi-voraz multi-arranque para diferentes valores de α
α = 0.05 (200 iter.)
α = 0.5 (200 iter.)
α = 0.95 (200 iter.)
Frecuencia
0
25
50
75
100
125
150
175
200
25000
27500
30000
32500
35000
37500
40000
42500
45000
Valor de la función objetivo (Costo total)

```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 20, 26, 32, 47]
  Costo total: 23976
  Total de iteraciones: 54
  Tiempo de ejecución: 0.24 segundos
  Estadísticas de las 54 iteraciones:
    Mejor: 23976
    Peor: 24198
    Promedio: 24120.98
    Desviación porcentual: 0.00%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 20, 26, 32, 47]
  Costo total: 23976
  Total de iteraciones: 65
  Tiempo de ejecución: 0.46 segundos
  Estadísticas de las 65 iteraciones:
    Mejor: 23976
    Peor: 30915
    Promedio: 25324.68
    Desviación porcentual: 0.00%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 20, 26, 32, 47]
  Costo total: 23976
  Total de iteraciones: 69
  Tiempo de ejecución: 0.59 segundos
  Estadísticas de las 69 iteraciones:
    Mejor: 23976
    Peor: 33563
    Promedio: 26579.71
    Desviación porcentual: 0.00%
```



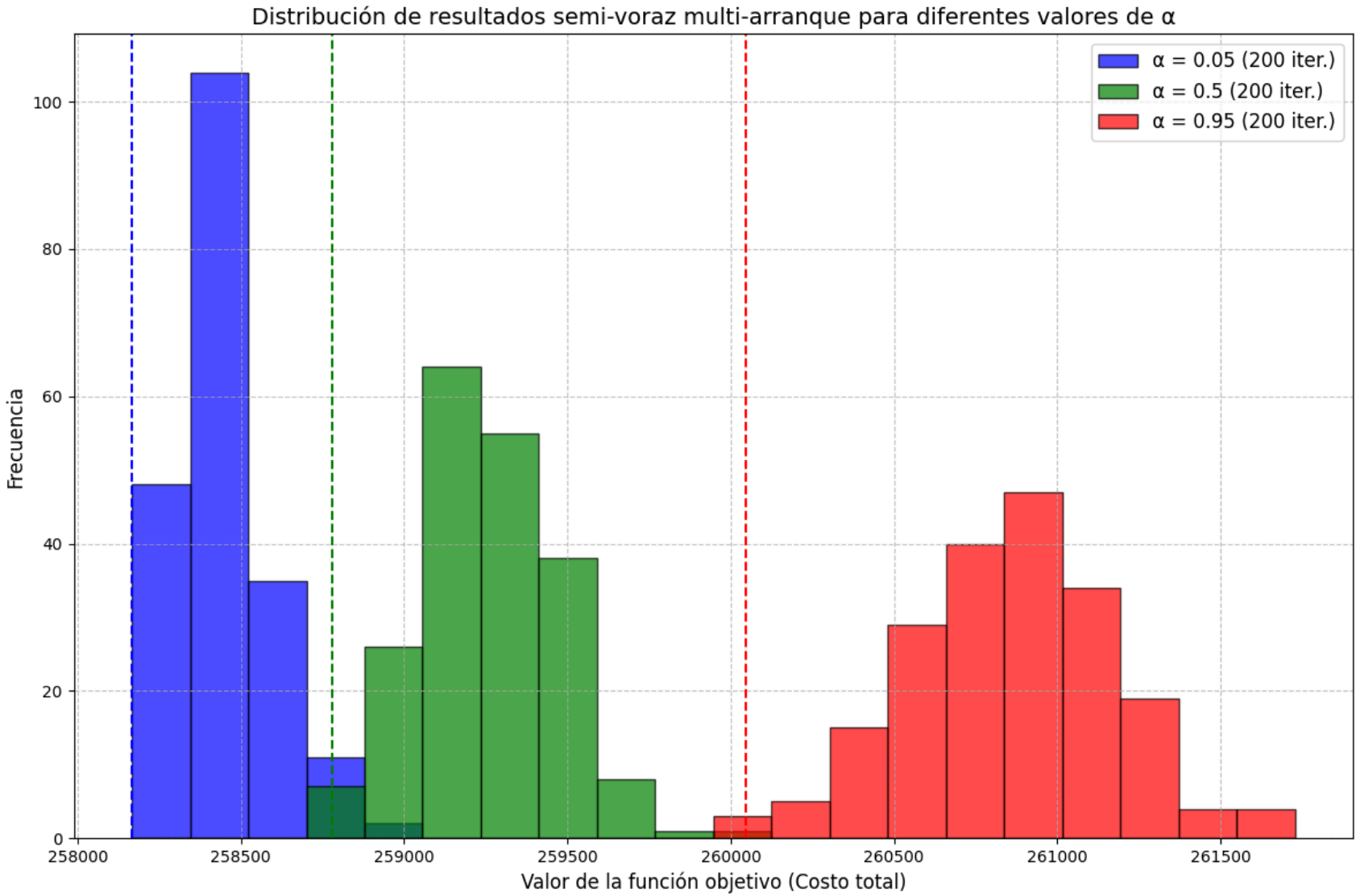

```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo GRASP con alpha = 0.05:
  Instalaciones seleccionadas: [9, 20, 26, 32, 47]
  Costo total: 23976
```

Instancia 6

In [15]:
```
ejecutar("UFLP-6.txt", 257964)
```

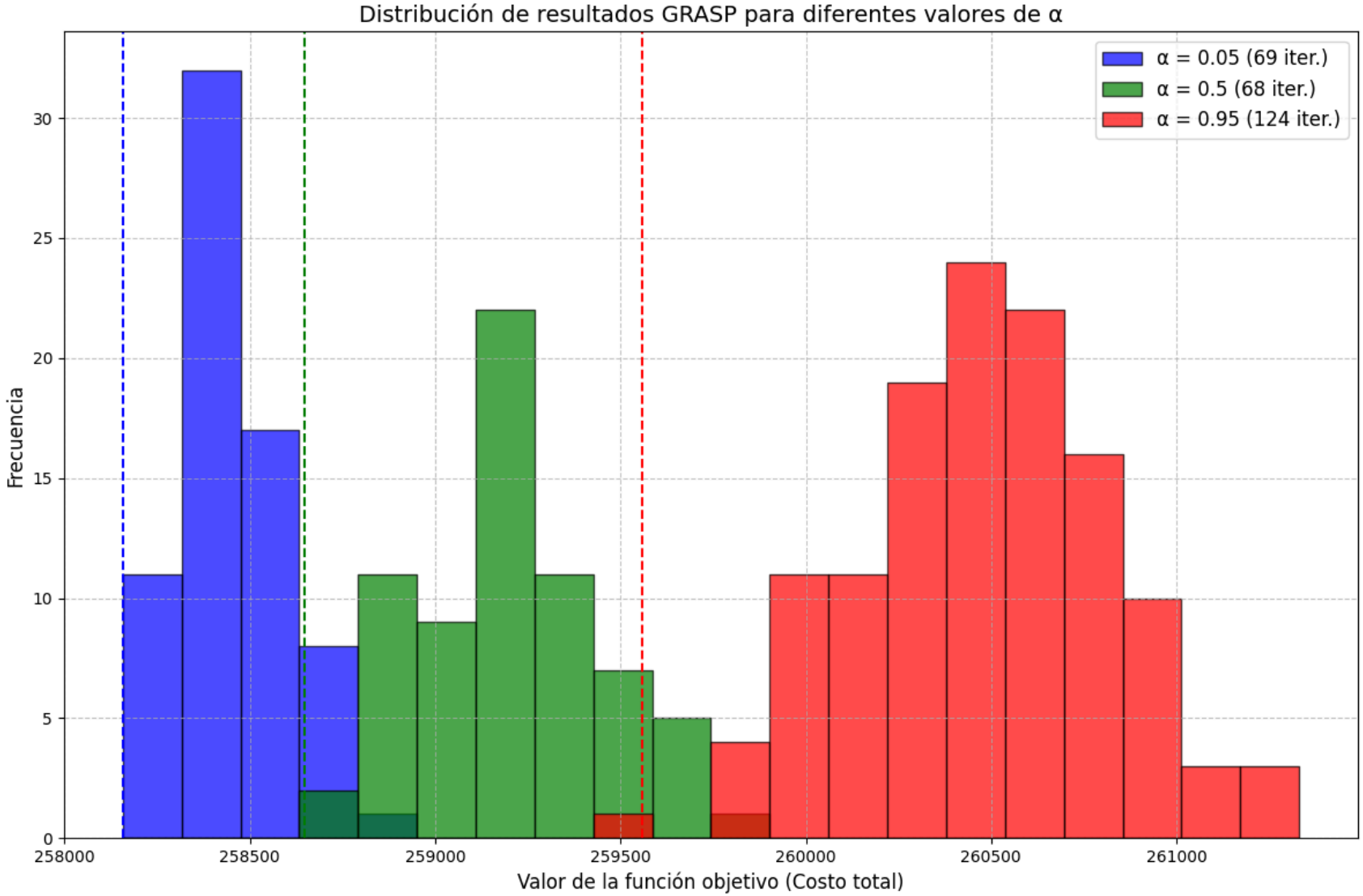
```
Problema con 250 ubicaciones potenciales y 250 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [10, 12, 19, 27, 30, 47, 51, 63, 64, 71, 80, 90, 97, 119, 123, 132, 137, 140, 145, 15
5, 166, 169, 170, 178, 187, 188, 203, 232, 234, 236, 238, 246, 249]
  Costo total: 258166
  Número de iteraciones: 200
  Tiempo de ejecución: 35.63 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 258166
    Peor: 258944
    Promedio: 258439.60
    Desviación porcentual: 0.08%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [10, 12, 16, 20, 47, 51, 54, 56, 64, 66, 80, 84, 90, 92, 97, 98, 106, 119, 130, 137, 1
45, 155, 161, 166, 170, 178, 184, 205, 210, 216, 229, 232, 234, 236, 238, 249]
  Costo total: 258780
  Número de iteraciones: 200
  Tiempo de ejecución: 40.40 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 258780
    Peor: 260051
    Promedio: 259257.77
    Desviación porcentual: 0.32%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [11, 12, 19, 27, 30, 46, 48, 50, 51, 53, 56, 60, 63, 66, 71, 86, 88, 89, 94, 110, 118,
123, 125, 126, 131, 132, 134, 136, 144, 145, 150, 155, 158, 160, 166, 167, 168, 177, 181, 193, 210, 213, 229, 231, 23
2, 234, 236, 244, 247, 249]
  Costo total: 260047
  Número de iteraciones: 200
  Tiempo de ejecución: 54.68 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 260047
```

```
Peor: 261728
Promedio: 260856.14
Desviación porcentual: 0.81%
```



```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [10, 11, 12, 17, 19, 27, 30, 47, 48, 51, 63, 64, 71, 86, 92, 97, 119, 130, 132, 145, 1
53, 161, 176, 178, 184, 197, 216, 229, 232, 234, 236, 249]
  Costo total: 258157
  Total de iteraciones: 69
  Tiempo de ejecución: 12.53 segundos
  Estadísticas de las 69 iteraciones:
    Mejor: 258157
    Peor: 258813
    Promedio: 258452.80
    Desviación porcentual: 0.07%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [5, 10, 12, 51, 63, 71, 73, 80, 86, 89, 97, 106, 116, 119, 123, 124, 131, 132, 137, 14
5, 155, 161, 162, 163, 167, 169, 184, 188, 206, 210, 232, 234, 236, 238, 242, 246, 248, 249]
  Costo total: 258646
  Total de iteraciones: 68
  Tiempo de ejecución: 14.57 segundos
  Estadísticas de las 68 iteraciones:
    Mejor: 258646
    Peor: 259813
    Promedio: 259200.57
    Desviación porcentual: 0.26%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [4, 10, 12, 15, 17, 25, 36, 51, 54, 61, 62, 66, 71, 89, 98, 101, 103, 104, 112, 113, 1
18, 119, 130, 131, 134, 137, 140, 142, 155, 160, 161, 163, 168, 170, 176, 177, 178, 179, 184, 187, 203, 222, 229, 23
2, 234, 235, 244, 249]
  Costo total: 259560
  Total de iteraciones: 124
  Tiempo de ejecución: 36.88 segundos
  Estadísticas de las 124 iteraciones:
    Mejor: 259560
    Peor: 261331
```

```
Promedio: 260469.10
Desviación porcentual: 0.62%
```



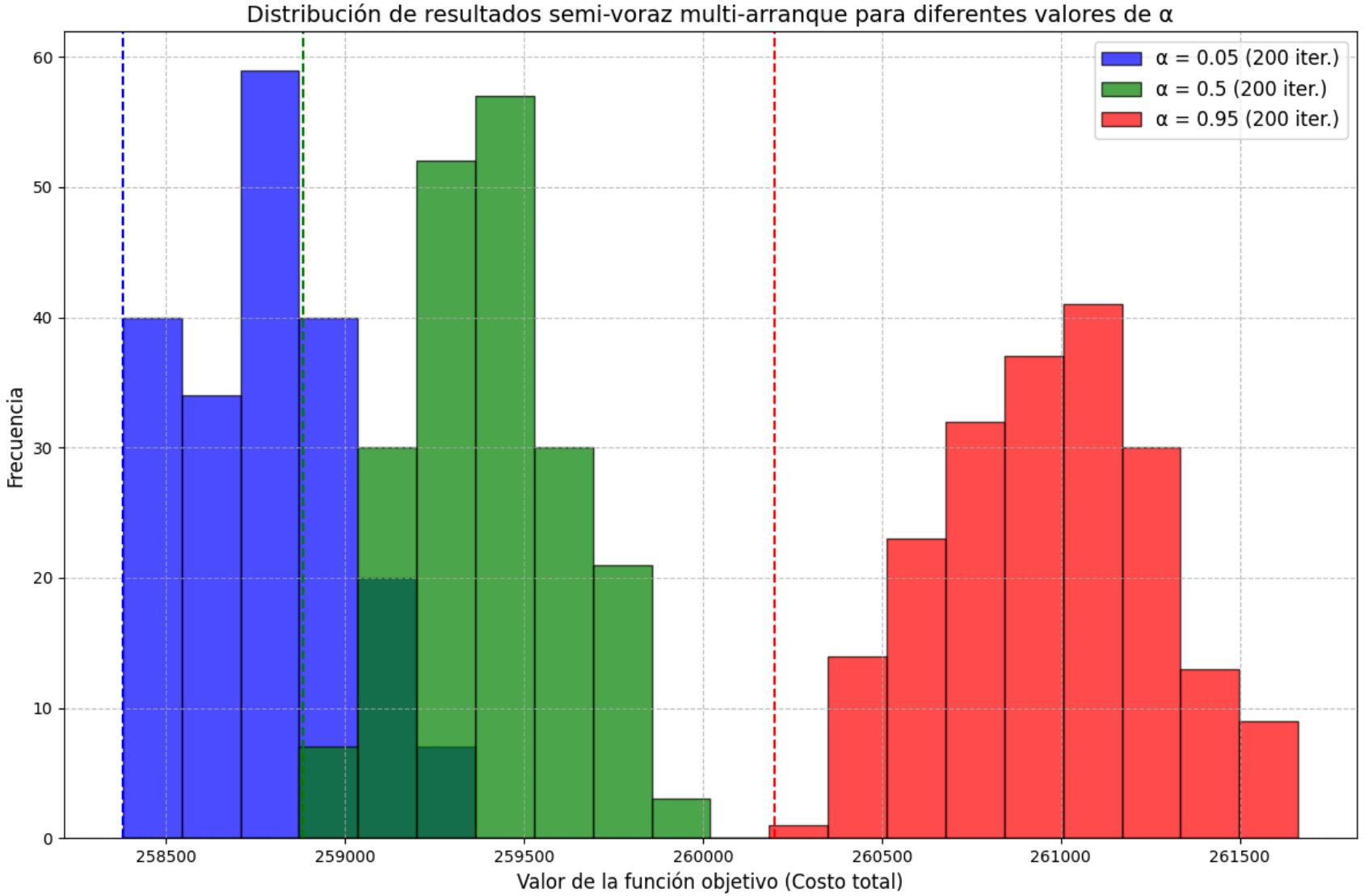
```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo GRASP con alpha = 0.05:
  Instalaciones seleccionadas: [10, 11, 12, 17, 19, 27, 30, 47, 48, 51, 63, 64, 71, 86, 92, 97, 119, 130, 132, 145, 1
53, 161, 176, 178, 184, 197, 216, 229, 232, 234, 236, 249]
  Costo total: 258157
```

Instancia 7

In [16]:
```
ejecutar("UFLP-7.txt", 257961)
```

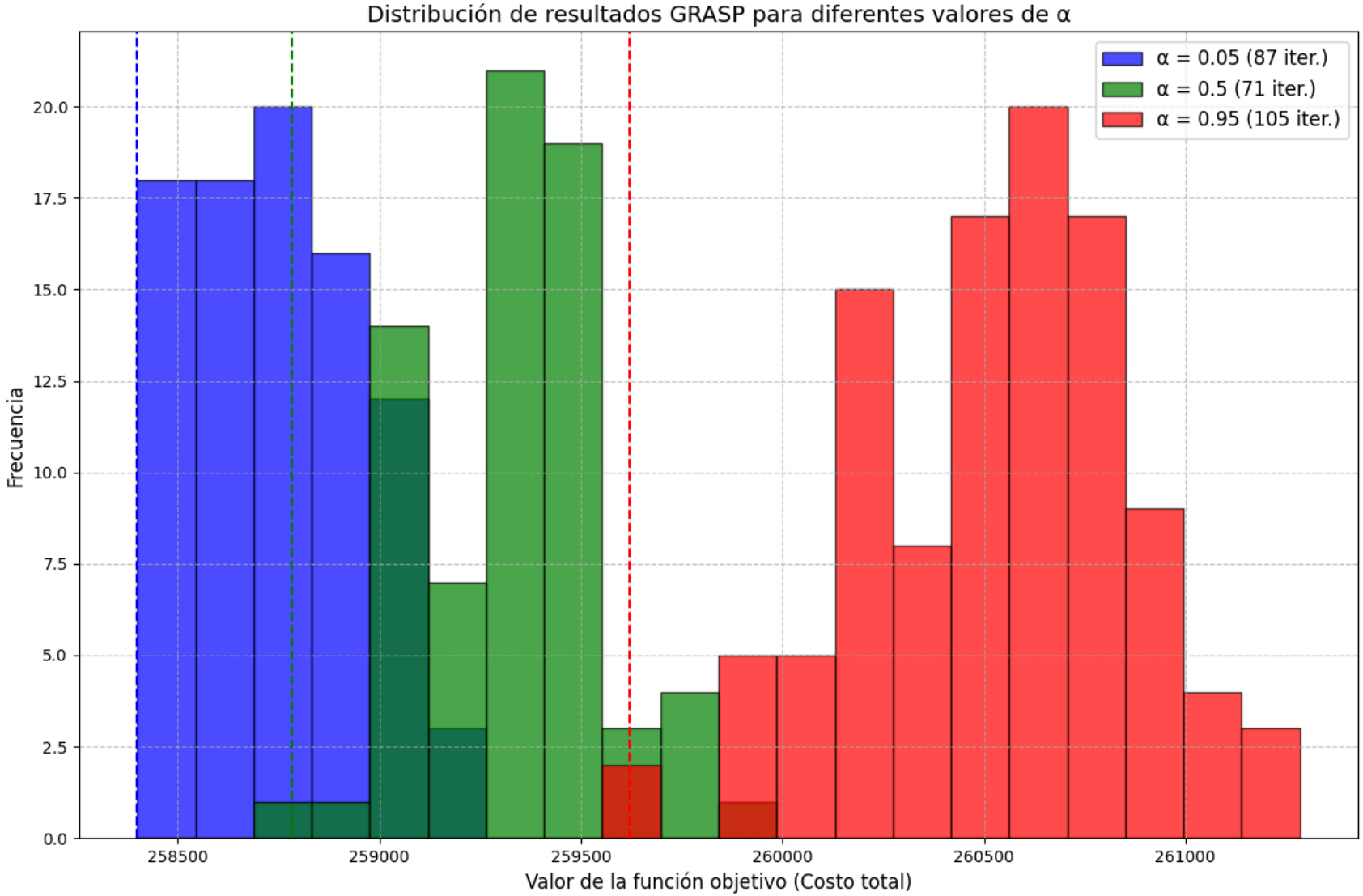
```
Problema con 250 ubicaciones potenciales y 250 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 4, 9, 20, 31, 32, 46, 54, 68, 93, 123, 137, 139, 141, 143, 148, 157, 159, 163, 16
6, 168, 169, 193, 195, 210, 213, 220, 226, 229, 237, 249]
  Costo total: 258379
  Número de iteraciones: 200
  Tiempo de ejecución: 37.70 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 258379
    Peor: 259271
    Promedio: 258767.28
    Desviación porcentual: 0.16%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 13, 20, 25, 31, 32, 36, 41, 50, 58, 86, 87, 93, 102, 104, 119, 123, 138, 139, 140,
143, 144, 150, 151, 157, 159, 163, 167, 169, 185, 188, 193, 208, 213, 221, 233, 247]
  Costo total: 258882
  Número de iteraciones: 200
  Tiempo de ejecución: 41.66 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 258882
    Peor: 259903
    Promedio: 259409.90
    Desviación porcentual: 0.36%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 9, 13, 16, 19, 20, 25, 29, 31, 34, 35, 36, 41, 43, 44, 49, 69, 70, 73, 76, 77, 81,
84, 90, 93, 96, 104, 108, 113, 116, 122, 129, 139, 143, 144, 157, 159, 163, 169, 179, 186, 189, 203, 215, 228, 237, 2
38, 248]
  Costo total: 260200
  Número de iteraciones: 200
  Tiempo de ejecución: 51.92 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 260200
```

```
Peor: 261663
Promedio: 260961.05
Desviación porcentual: 0.87%
```



```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 9, 20, 31, 32, 46, 54, 57, 68, 115, 123, 137, 139, 141, 143, 144, 148, 157, 159, 1
63, 166, 168, 169, 193, 195, 208, 210, 213, 226, 229, 237, 249]
  Costo total: 258399
  Total de iteraciones: 87
  Tiempo de ejecución: 16.81 segundos
  Estadísticas de las 87 iteraciones:
    Mejor: 258399
    Peor: 259183
    Promedio: 258737.44
    Desviación porcentual: 0.17%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 6, 9, 14, 19, 20, 31, 32, 35, 44, 50, 54, 66, 91, 93, 102, 118, 123, 143, 157, 15
9, 163, 166, 167, 168, 169, 175, 179, 189, 193, 197, 208, 230, 231, 247]
  Costo total: 258784
  Total de iteraciones: 71
  Tiempo de ejecución: 17.57 segundos
  Estadísticas de las 71 iteraciones:
    Mejor: 258784
    Peor: 259899
    Promedio: 259329.38
    Desviación porcentual: 0.32%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 14, 18, 21, 31, 35, 37, 42, 47, 54, 55, 69, 70, 71, 73, 77, 82, 102, 104, 107, 11
0, 123, 124, 130, 143, 144, 146, 155, 159, 160, 163, 167, 169, 182, 189, 193, 208, 215, 227, 231, 233, 237, 238, 246,
247]
  Costo total: 259621
  Total de iteraciones: 105
  Tiempo de ejecución: 33.09 segundos
  Estadísticas de las 105 iteraciones:
    Mejor: 259621
    Peor: 261284
```

```
Promedio: 260523.10
Desviación porcentual: 0.64%
```



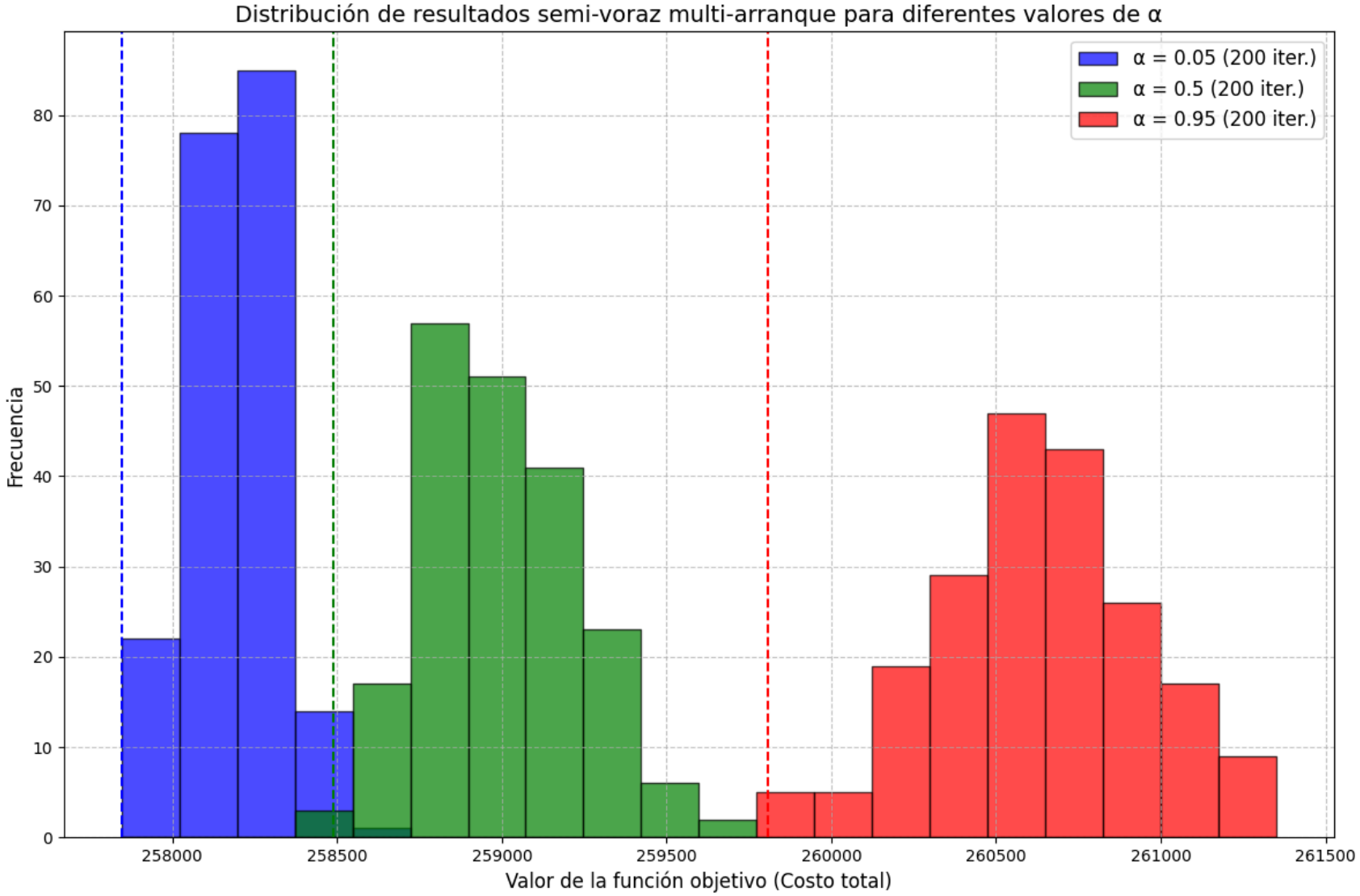
```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo Semi-voraz multi-arranque con alpha = 0.05:
  Instalaciones seleccionadas: [3, 4, 9, 20, 31, 32, 46, 54, 68, 93, 123, 137, 139, 141, 143, 148, 157, 159, 163, 16
6, 168, 169, 193, 195, 210, 213, 220, 226, 229, 237, 249]
  Costo total: 258379
```

Instancia 8

In [17]:
```
ejecutar("UFLP-8.txt", 257626)
```

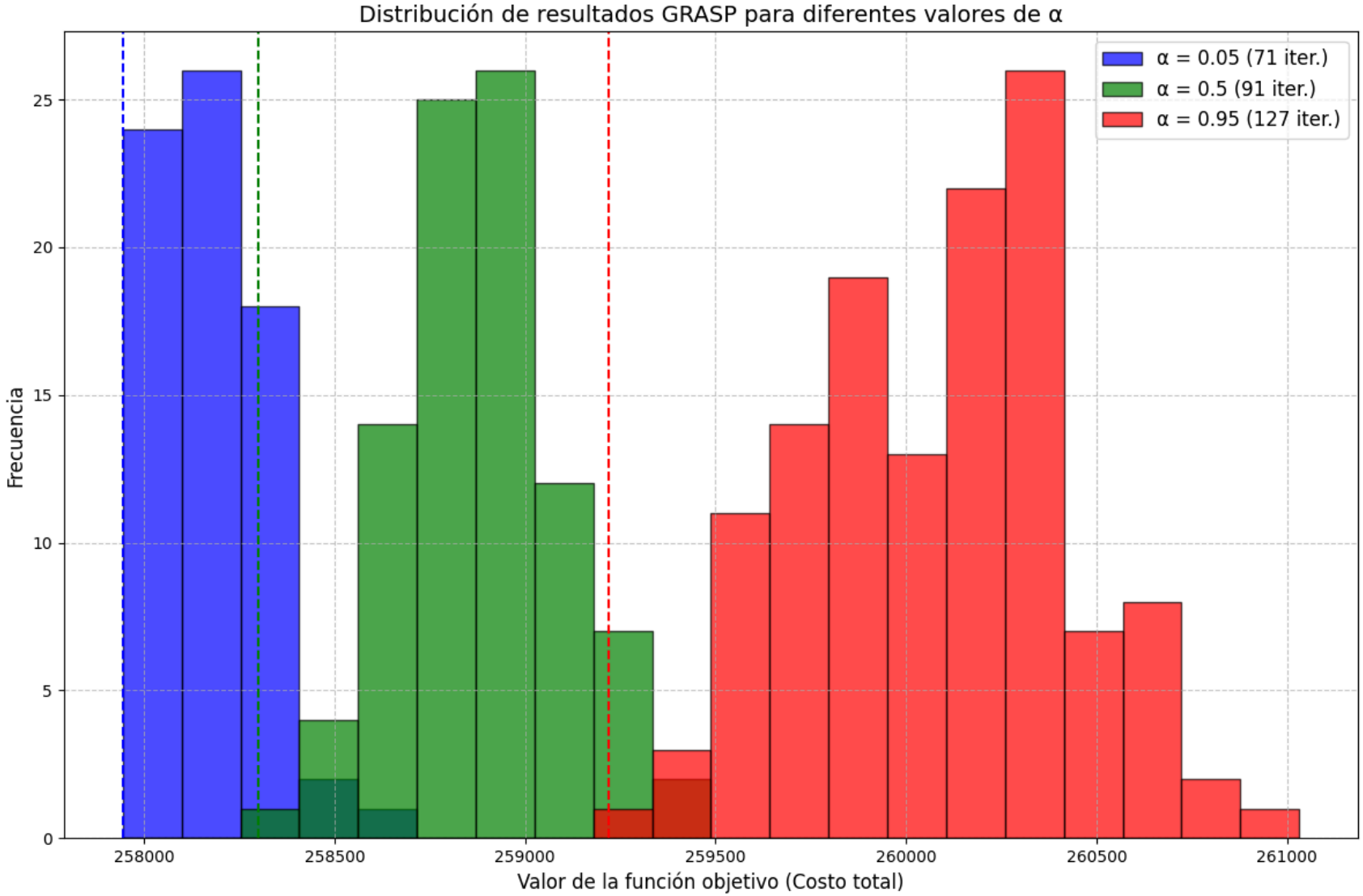
```
Problema con 250 ubicaciones potenciales y 250 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [33, 34, 37, 52, 55, 60, 67, 69, 71, 73, 91, 111, 131, 134, 137, 138, 139, 142, 144, 1
46, 166, 172, 174, 176, 177, 179, 182, 197, 213, 216, 220, 229]
  Costo total: 257846
  Número de iteraciones: 200
  Tiempo de ejecución: 39.32 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 257846
    Peor: 258570
    Promedio: 258195.88
    Desviación porcentual: 0.09%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 27, 34, 37, 38, 49, 52, 60, 64, 67, 74, 92, 99, 103, 106, 107, 110, 120, 133, 139,
146, 159, 160, 174, 176, 179, 190, 199, 200, 208, 210, 213, 223, 229, 233, 243, 248]
  Costo total: 258487
  Número de iteraciones: 200
  Tiempo de ejecución: 44.36 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 258487
    Peor: 259623
    Promedio: 258994.96
    Desviación porcentual: 0.33%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [8, 9, 14, 30, 65, 69, 74, 81, 86, 94, 101, 121, 128, 129, 132, 134, 135, 136, 142, 14
6, 147, 148, 150, 166, 174, 175, 183, 188, 190, 198, 199, 204, 214, 219, 220, 221, 223, 225, 228, 229, 234, 238, 240]
  Costo total: 259808
  Número de iteraciones: 200
  Tiempo de ejecución: 56.98 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 259808
    Peor: 261350
```

```
Promedio: 260630.95
Desviación porcentual: 0.85%
```



```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [37, 52, 55, 60, 64, 67, 69, 71, 94, 131, 134, 137, 138, 139, 144, 146, 149, 162, 166,
172, 174, 176, 177, 179, 199, 213, 216, 220, 229, 230, 248]
  Costo total: 257944
  Total de iteraciones: 71
  Tiempo de ejecución: 13.33 segundos
  Estadísticas de las 71 iteraciones:
    Mejor: 257944
    Peor: 258621
    Promedio: 258180.87
    Desviación porcentual: 0.12%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 16, 34, 37, 38, 40, 55, 60, 64, 71, 91, 95, 103, 110, 111, 126, 132, 139, 140, 14
4, 148, 149, 162, 166, 172, 174, 177, 179, 186, 190, 197, 199, 209, 216, 229, 239]
  Costo total: 258301
  Total de iteraciones: 91
  Tiempo de ejecución: 20.12 segundos
  Estadísticas de las 91 iteraciones:
    Mejor: 258301
    Peor: 259417
    Promedio: 258880.20
    Desviación porcentual: 0.26%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 10, 24, 34, 37, 52, 60, 67, 71, 75, 82, 86, 103, 107, 111, 122, 128, 136, 137, 13
9, 142, 144, 156, 166, 168, 170, 172, 179, 183, 186, 189, 190, 191, 196, 204, 208, 210, 213, 219, 221, 223, 227, 232,
233, 241]
  Costo total: 259219
  Total de iteraciones: 127
  Tiempo de ejecución: 36.86 segundos
  Estadísticas de las 127 iteraciones:
    Mejor: 259219
    Peor: 261032
```

```
Promedio: 260083.88
Desviación porcentual: 0.62%
```



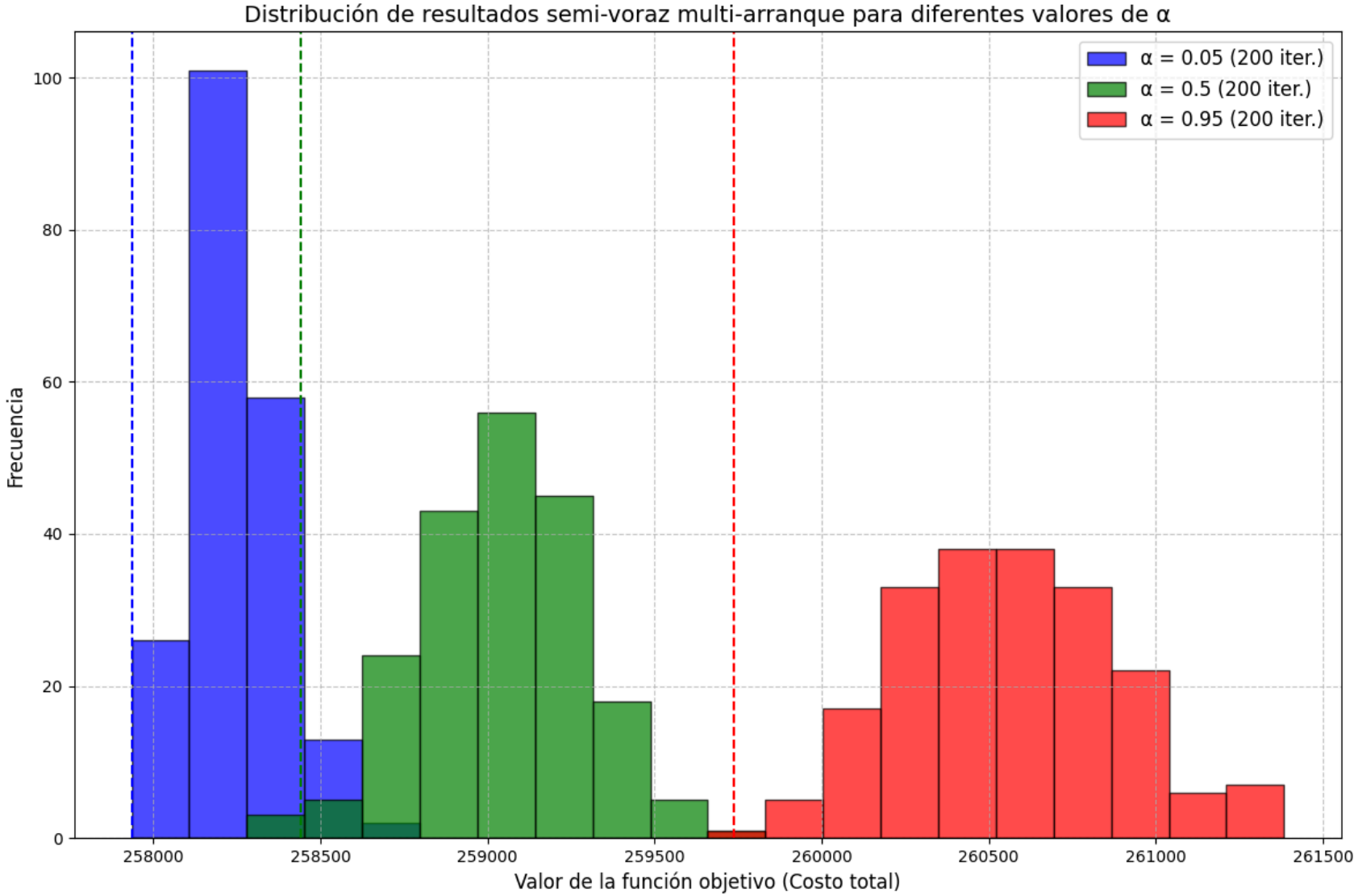
```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo Semi-voraz multi-arranque con alpha = 0.05:
  Instalaciones seleccionadas: [33, 34, 37, 52, 55, 60, 67, 69, 71, 73, 91, 111, 131, 134, 137, 138, 139, 142, 144, 1
46, 166, 172, 174, 176, 177, 179, 182, 197, 213, 216, 220, 229]
  Costo total: 257846
```

Instancia 9

In [18]:
```
ejecutar("UFLP-9.txt", 257573)
```

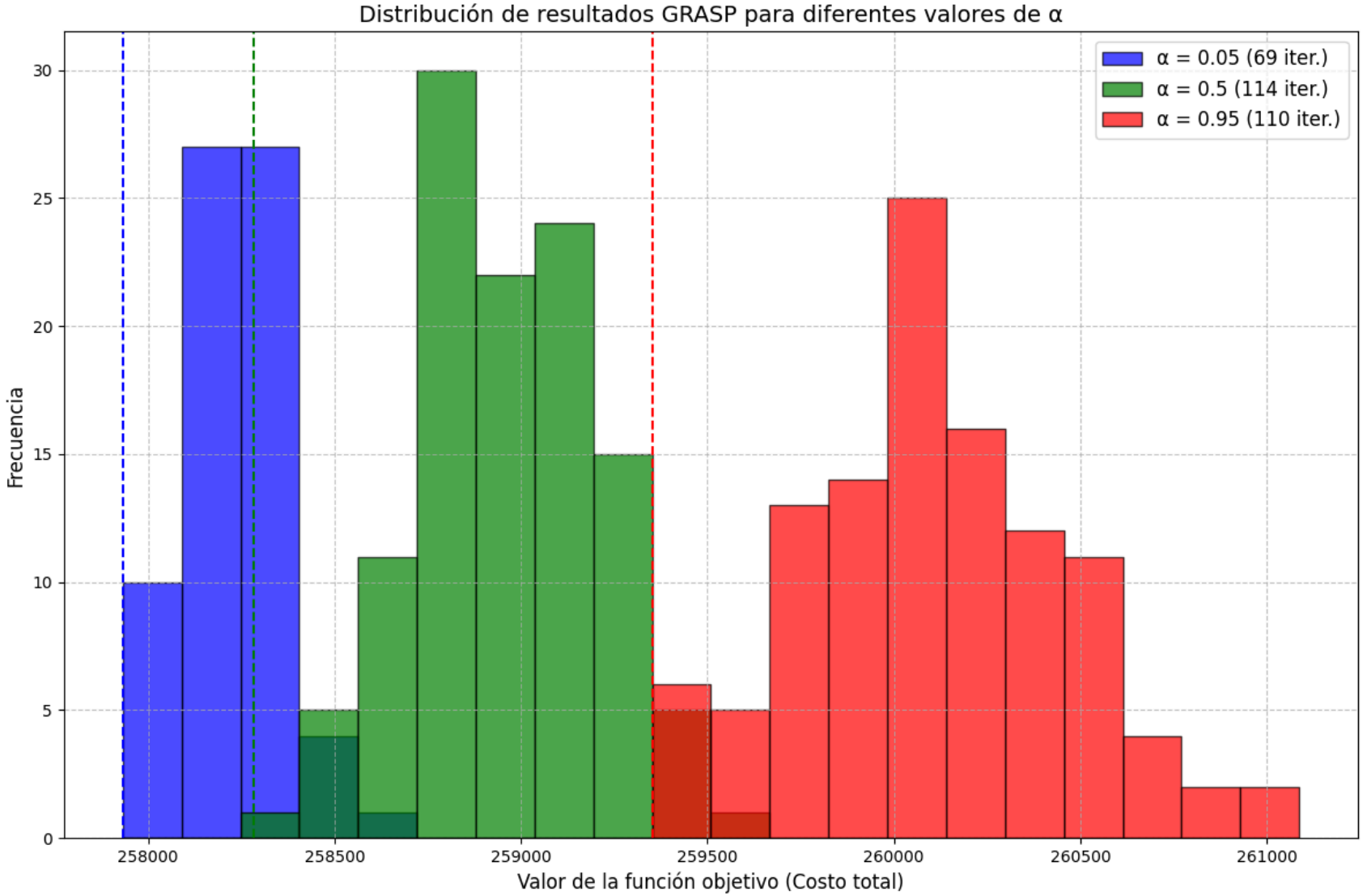
```
Problema con 250 ubicaciones potenciales y 250 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 17, 24, 25, 40, 43, 46, 52, 74, 76, 77, 87, 88, 95, 96, 98, 101, 120, 130, 132, 13
9, 142, 147, 152, 154, 160, 161, 166, 184, 191, 234, 241, 250]
  Costo total: 257935
  Número de iteraciones: 200
  Tiempo de ejecución: 36.78 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 257935
    Peor: 258675
    Promedio: 258247.66
    Desviación porcentual: 0.14%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 12, 25, 43, 46, 52, 54, 77, 95, 96, 98, 99, 101, 106, 113, 114, 120, 126, 130, 13
1, 138, 139, 140, 147, 149, 154, 160, 164, 174, 184, 190, 229, 241, 249, 250]
  Costo total: 258440
  Número de iteraciones: 200
  Tiempo de ejecución: 43.86 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 258440
    Peor: 259785
    Promedio: 259045.97
    Desviación porcentual: 0.34%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 22, 24, 42, 43, 46, 49, 61, 65, 72, 81, 87, 89, 91, 92, 96, 101, 109, 119, 122, 12
4, 128, 130, 154, 155, 160, 163, 164, 165, 166, 173, 174, 179, 180, 182, 206, 212, 218, 221, 224, 231, 239, 245, 249]
  Costo total: 259736
  Número de iteraciones: 200
  Tiempo de ejecución: 53.32 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 259736
    Peor: 261383
```

```
Promedio: 260554.74
Desviación porcentual: 0.84%
```



```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [4, 9, 17, 24, 25, 40, 43, 46, 52, 74, 76, 77, 87, 88, 95, 96, 98, 101, 113, 120, 130,
139, 142, 147, 154, 155, 160, 161, 166, 184, 191, 234, 241, 250]
  Costo total: 257932
  Total de iteraciones: 69
  Tiempo de ejecución: 13.45 segundos
  Estadísticas de las 69 iteraciones:
    Mejor: 257932
    Peor: 258606
    Promedio: 258228.09
    Desviación porcentual: 0.14%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [4, 21, 24, 43, 44, 46, 52, 58, 77, 85, 87, 88, 95, 96, 99, 100, 101, 103, 114, 116, 1
20, 128, 130, 131, 139, 152, 154, 160, 166, 187, 211, 212, 224, 241]
  Costo total: 258284
  Total de iteraciones: 114
  Tiempo de ejecución: 24.85 segundos
  Estadísticas de las 114 iteraciones:
    Mejor: 258284
    Peor: 259626
    Promedio: 258964.06
    Desviación porcentual: 0.28%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [1, 2, 3, 24, 25, 37, 40, 42, 44, 46, 50, 52, 64, 71, 85, 86, 87, 92, 94, 95, 101, 10
7, 110, 113, 114, 118, 126, 129, 130, 133, 139, 140, 147, 149, 154, 158, 161, 184, 190, 191, 218, 219, 220, 225, 234,
237, 241, 245]
  Costo total: 259354
  Total de iteraciones: 110
  Tiempo de ejecución: 30.80 segundos
  Estadísticas de las 110 iteraciones:
    Mejor: 259354
    Peor: 261087
```

```
Promedio: 260109.75
Desviación porcentual: 0.69%
```



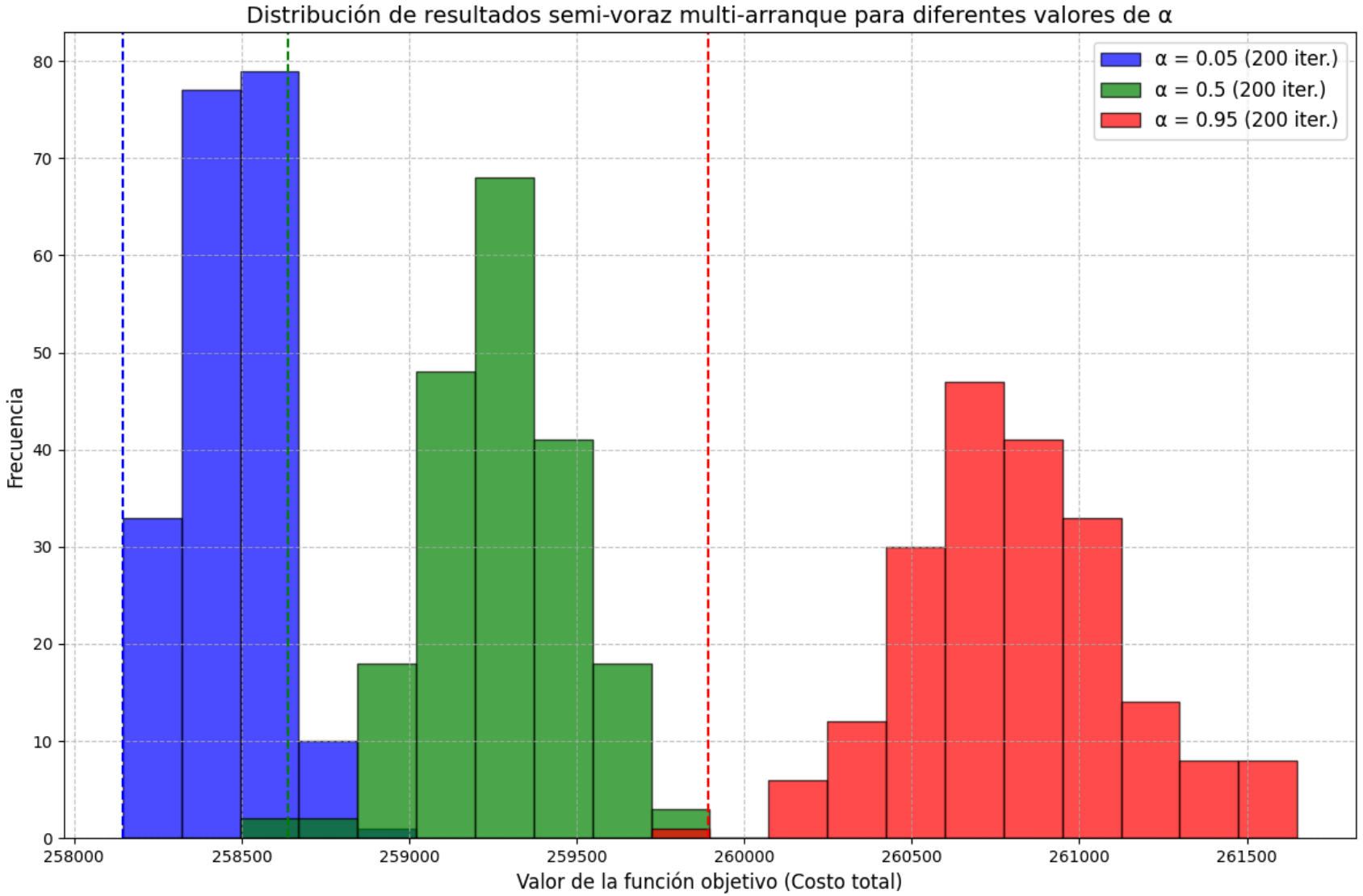
```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo GRASP con alpha = 0.05:
  Instalaciones seleccionadas: [4, 9, 17, 24, 25, 40, 43, 46, 52, 74, 76, 77, 87, 88, 95, 96, 98, 101, 113, 120, 130,
139, 142, 147, 154, 155, 160, 161, 166, 184, 191, 234, 241, 250]
  Costo total: 257932
```

Instancia 10

In [19]:
```
ejecutar("UFLP-10.txt",  257896)
```

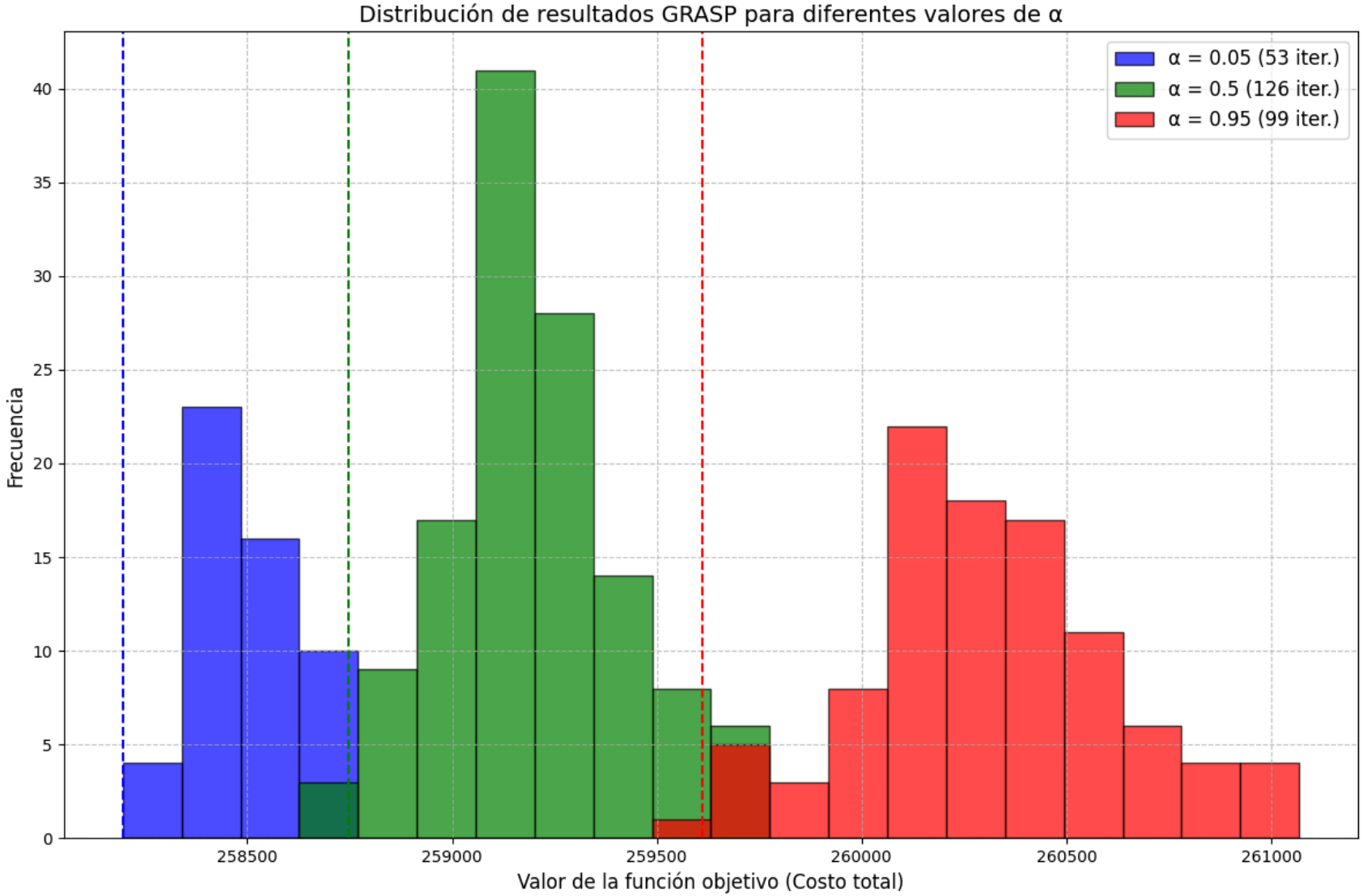
```
Problema con 250 ubicaciones potenciales y 250 puntos de demanda

================================================================================
PARTE 1: HEURÍSTICA SEMI-VORAZ MULTI-ARRANQUE
================================================================================

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 18, 24, 38, 46, 49, 53, 59, 60, 62, 77, 82, 87, 89, 98, 114, 118, 120, 122, 124, 1
32, 133, 137, 166, 172, 204, 206, 209, 212, 223, 228, 230, 234]
  Costo total: 258145
  Número de iteraciones: 200
  Tiempo de ejecución: 37.56 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 258145
    Peor: 258894
    Promedio: 258474.54
    Desviación porcentual: 0.10%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 18, 26, 33, 47, 49, 60, 76, 86, 90, 98, 100, 104, 110, 111, 118, 124, 125, 133, 13
7, 145, 150, 159, 161, 167, 172, 189, 194, 200, 204, 207, 209, 212, 213, 227, 228, 230, 231, 240, 245, 250]
  Costo total: 258638
  Número de iteraciones: 200
  Tiempo de ejecución: 43.59 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 258638
    Peor: 259870
    Promedio: 259276.57
    Desviación porcentual: 0.29%

Ejecutando algoritmo semi-voraz multi-arranque con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 13, 14, 15, 23, 24, 31, 36, 40, 45, 49, 69, 83, 84, 87, 89, 100, 115, 122, 127, 13
3, 139, 142, 149, 155, 158, 161, 166, 167, 172, 176, 179, 182, 196, 200, 202, 204, 207, 209, 212, 214, 225, 226, 228,
230, 245, 250]
  Costo total: 259893
  Número de iteraciones: 200
  Tiempo de ejecución: 54.30 segundos
  Estadísticas de las 200 iteraciones:
    Mejor: 259893
```

```
Peor: 261650
Promedio: 260811.01
Desviación porcentual: 0.77%
```



```
================================================================================
PARTE 2: ALGORITMO GRASP
================================================================================

Ejecutando algoritmo GRASP con alpha = 0.05:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [9, 18, 33, 38, 43, 49, 50, 51, 53, 62, 79, 87, 89, 111, 112, 114, 118, 122, 132, 133,
160, 166, 172, 194, 200, 204, 206, 207, 209, 212, 214, 230, 250]
  Costo total: 258196
  Total de iteraciones: 53
  Tiempo de ejecución: 10.24 segundos
  Estadísticas de las 53 iteraciones:
    Mejor: 258196
    Peor: 258740
    Promedio: 258490.72
    Desviación porcentual: 0.12%

Ejecutando algoritmo GRASP con alpha = 0.5:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [18, 28, 33, 36, 47, 49, 50, 51, 59, 60, 70, 76, 86, 89, 90, 99, 111, 115, 122, 126, 1
30, 133, 137, 149, 156, 158, 166, 169, 172, 200, 204, 207, 208, 209, 217, 223, 230, 236, 249]
  Costo total: 258748
  Total de iteraciones: 126
  Tiempo de ejecución: 28.43 segundos
  Estadísticas de las 126 iteraciones:
    Mejor: 258748
    Peor: 259762
    Promedio: 259199.82
    Desviación porcentual: 0.33%

Ejecutando algoritmo GRASP con alpha = 0.95:
  Mejor solución encontrada:
  Instalaciones seleccionadas: [3, 18, 22, 33, 47, 51, 53, 60, 62, 63, 77, 84, 86, 98, 100, 104, 110, 112, 124, 130,
133, 139, 140, 145, 147, 150, 153, 156, 157, 160, 166, 172, 178, 184, 194, 200, 204, 207, 209, 212, 221, 227, 235, 24
0, 244, 246, 250]
  Costo total: 259611
  Total de iteraciones: 99
  Tiempo de ejecución: 28.73 segundos
  Estadísticas de las 99 iteraciones:
    Mejor: 259611
    Peor: 261068
```

```
Promedio: 260308.17
Desviación porcentual: 0.66%
```



```
================================================================================
MEJOR SOLUCIÓN GLOBAL
================================================================================
La mejor solución global fue encontrada por el algoritmo Semi-voraz multi-arranque con alpha = 0.05:
  Instalaciones seleccionadas: [9, 18, 24, 38, 46, 49, 53, 59, 60, 62, 77, 82, 87, 89, 98, 114, 118, 120, 122, 124, 1
32, 133, 137, 166, 172, 204, 206, 209, 212, 223, 228, 230, 234]
  Costo total: 258145
```