AG3 - Actividad Guiada 3 - Simulación de Colonia de Hormigas (ACO)

Nombre: Carlos Javier Bravo Intriago

Link: https://colab.research.google.com/drive/1Yd5cj-AmDNnWgQpIo7UXIurUMpJbt1ZA?usp=sharing

Github: https://github.com/carlosbravo1408/03MIAR-Algoritmos-de-Optimizacion-2025/tree/main/AG3

Carga de librerias

In [1]:
```
!pip install requests
!pip install tabulate>=0.9 networkx>=3.0
!pip install tsplib95 --no-deps
!pip install deprecated
```

```
Requirement already satisfied: requests in /home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/python3.
10/site-packages (2.32.5)
Requirement already satisfied: charset_normalizer<4,>=2 in /home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.v
env/lib/python3.10/site-packages (from requests) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/pyth
on3.10/site-packages (from requests) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/li
b/python3.10/site-packages (from requests) (2.6.3)
Requirement already satisfied: certifi>=2017.4.17 in /home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/li
b/python3.10/site-packages (from requests) (2026.1.4)

[notice] A new release of pip is available: 24.3.1 -> 26.0.1
[notice] To update, run: pip install --upgrade pip

[notice] A new release of pip is available: 24.3.1 -> 26.0.1
[notice] To update, run: pip install --upgrade pip
Requirement already satisfied: tsplib95 in /home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/python3.
10/site-packages (0.7.1)

[notice] A new release of pip is available: 24.3.1 -> 26.0.1
[notice] To update, run: pip install --upgrade pip
Requirement already satisfied: deprecated in /home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/python
3.10/site-packages (1.2.18)
Requirement already satisfied: wrapt<2,>=1.10 in /home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/py
thon3.10/site-packages (from deprecated) (1.17.3)

[notice] A new release of pip is available: 24.3.1 -> 26.0.1
[notice] To update, run: pip install --upgrade pip
```

Importe de librerias

In [2]:
```
from typing import Union, Tuple, List, Literal, Dict, Any
import random
import gzip
import shutil
import os
import urllib.request
from copy import deepcopy

import tsplib95
```

Métodos Auxiliares y algunas definiciones necesarias

In [3]:
```
NumericType = Union[int, float]
NodeType = int
EdgeType = Tuple[NodeType, NodeType]
SolutionType = List[NodeType]
TSProblemType = tsplib95.models.Problem

def download_tsp_file(url: str, filename: str) -> None:
    # Descarga como archivo temporal
    local_path, headers = urllib.request.urlretrieve(url, filename + ".temp")
    mime_type = headers.get_content_type()
    # Descomprimir si es formato gzip
```

```
    if "gzip" in mime_type or local_path.endswith(".gz"):
        with gzip.open(local_path, 'rb') as f_in:
            with open(filename, 'wb') as f_out:
                shutil.copyfileobj(f_in, f_out)
        os.remove(local_path)
    # Caso contrario renombra el archivo temporal
    else:
        os.replace(local_path, filename)
```

In [4]:
```
#DATOS DEL PROBLEMA
# Matriz Adyacencia swiss42 problem (Staedte Schweiz/Fricker)
file = "swiss42.tsp"
download_tsp_file("https://raw.githubusercontent.com/mastqe/tsplib/refs/heads/master/swiss42.tsp", file)
tsp_problem: tsplib95.models.Problem = tsplib95.load(file)

# Lo siguiente NO es una buena práctica de programacion y hay gente que
# considera sucia, pero permite optimizar algo más el código. Se inyecta la
# lista de nodos en el objeto tsp_problem, con ello se evita crear una nueva
# lista cada vez que se invoca a `get_nodes()`
tsp_problem.cached_nodes = list(tsp_problem.get_nodes())

#Nodos
nodes = tsp_problem.cached_nodes
```

Funcionas básicas

In [5]:
```
def distancia(a: int, b: int, problem: TSProblemType) -> NumericType:
    """
    Devuelve la distancia entre dos nodos
    :param a: Nodo a
    :param b: Nodo b
    :param problem: Instancia de tsplib95.models.Problem
    :return:
    """
    return problem.get_weight(a, b)

def distancia_total(solucion: List[int], problem: TSProblemType) -> NumericType:
    """
    Devuelve la distancia total de una trayectoria/solución
    :param solucion: Lista de Nodos
    :param problem: Instancia de tsplib95.models.Problem
    :return:
    """
    distancia_total = 0
    n = len(solucion)
    for i in range(len(solucion)):
        distancia_total += distancia(solucion[i], solucion[(i + 1)%n], problem)
    return distancia_total

def crear_solucion(nodos: List[NodeType]) -> SolutionType:
    """
    Se genera una solución aleatoria con comienzo en el nodo 0
    :param nodos: Lista de nodos
    :return: Lista de nodos en posiciones aleatorias sin repetición
    """
    solucion = [nodos[0]]
    for _ in nodos[1:]:
        solucion = solucion + [
            random.choice(list(set(nodos) - {nodos[0]} - set(solucion)))]
    return solucion
```

Algoritmo de colonia de hormigas

La función Add_Nodo selecciona al azar un nodo con probabilidad uniforme. Para ser más eficiente debería seleccionar el próximo nodo siguiendo la probabilidad correspondiente a la ecuación:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\nu_{ij}]^\beta}{\sum_{l \in J_i^k} [\tau_{il}(t)]^\alpha [\nu_{il}]^\beta}, \text{ si } j \in J_i^k$$

$$p_{ij}^k(t) = 0, \text{ si } j \notin J_i^k$$

In [6]:
```
def add_node(
        problem: TSProblemType,
        H: SolutionType,
```

```
        T: List[List[NumericType]]
) -> NumericType:
    """
    Mejora: Establecer una función de probabilidad para añadir un nuevo nodo
    dependiendo de los nodos más cercanos y de las feromonas depositadas
    :param problem:
    :param H:
    :param T:
    :return:
    """
    nodos = problem.cached_nodes
    return random.choice(list(set(range(1, len(nodos))) - set(H)))

def increase_pheromone(
        problem: TSProblemType,
        T: List[List[NumericType]],
        H: SolutionType,
        q_factor: float = 1000.0
) -> List[List[NumericType]]:
    """
    Incrementa según la calidad de la solución. Añadir una cantidad
    inversamente proporcional a la distancia total
    :param problem:
    :param T:
    :param H:
    :param q_factor:
    :return:
    """
    n = len(H)
    distancia_recorrida = distancia_total(H, problem)
    delta = q_factor / distancia_recorrida
    for i in range(n):
        T[H[i]][H[(i + 1)%n]] += delta
    return T


def evaporate_pheromones(T: List[List[NodeType]]) -> List[List[NumericType]]:
    """
    Evapora 0.3 el valor de la feromona, sin que baje de 1\n
    Mejora: Podemos elegir diferentes funciones de evaporación dependiendo
    de la cantidad actual y de la suma total de feromonas depositadas,...
    :param T:
    :return:
    """
    T = [
        [max(T[i][j] - 0.3, 1) for i in range(len(nodes))]
        for j in range(len(nodes))
    ]
    return T

def init_pheromones(problem: TSProblemType) -> List[List[NumericType]]:
    """
    Inicializa las aristas con una cantidad inicial de feromonas:1\n
    Mejora: inicializar con valores diferentes dependiendo diferentes criterios
    :param n:
    :return:
    """
    m = problem.dimension
    return [[1 for _ in range(m)] for _ in range(m)]
```

Mejoras propuestas

In [7]:

```
def add_node_uniform(
        problem: TSProblemType,
        H: SolutionType,
        T: List[List[NumericType]],
        alpha: float = 1.0,
        beta: float = 2.0
) -> NumericType:
    """
    Selecciona el siguiente nodo usando la Regla de Transición Aleatoria
    Proporcional del Ant System (Dorigo et al., 1996).

    Probabilidad P_ij = (tau_ij^alpha * eta_ij^beta) / Sum(...)
    :param problem:
    :param H:
    :param T:
    :param alpha:
    :param beta:
```

```
    :return:
    """
    current_node = H[-1]
    n_nodes = len(T)

    candidates = [n for n in range(n_nodes) if n not in H]
    if not candidates:
        return None

    weights = []
    for node_j in candidates:
        tau = T[current_node][node_j]
        dist = problem.get_weight(current_node, node_j)
        eta = 1.0 / (dist if dist > 1e-10 else 1e-10)
        w = (tau ** alpha) * (eta ** beta)
        weights.append(w)

    return random.choices(candidates, weights=weights, k=1)[0]

def evaporate_pheromones_exponential_decay(
        T: List[List[NumericType]],
        rho: float = 0.1,
        tau_min: float = 1e-10
) -> List[List[NumericType]]:
    """
    Implementa la evaporación global del Ant System (Dorigo et al., 1996).
    *Se aplica a TODA la matriz T.*

    :param T: Matriz de feromonas
    :param rho: Coeficiente de evaporación (0 < rho <= 1).
                rho=0.5 implica que el 50% de la información se olvida en
                cada ciclo.
    :param tau_min: Epsilon para evitar que la feromona llegue a 0 absoluto
                    (lo cual causaría divisiones por cero en la probabilidad).
    """
    decay_factor = 1.0 - rho
    n = len(T)
    for i in range(n):
        for j in range(n):
            T[i][j] = max(T[i][j]*decay_factor, tau_min)
    return T

def init_pheromones_adaptative(problem: TSProblemType) -> List[List[NumericType]]:
    """
    Inicialización de feromonas para Ant System (Dorigo et al., 1996).
    Calcula tau_0 = 1 / (m * path_cost)

    :param problem: Instancia del problema TSP.
    """
    nodes = problem.cached_nodes
    m = problem.dimension

    current_node = nodes[0]
    unvisited = set(nodes[1:])
    path_cost = 0.0

    while unvisited:
        next_node = min(unvisited, key=lambda x: problem.get_weight(current_node, x))
        path_cost += problem.get_weight(current_node, next_node)
        unvisited.remove(next_node)
        current_node = next_node
    path_cost += problem.get_weight(current_node, nodes[0])
    tau0 = 1.0 / (m * path_cost) # Variante menos ruidosa, la original es n_hormigas / path_cost
    return [[tau0 for _ in range(m)] for _ in range(m)]

init_pheromones_strategies = {
    "base": init_pheromones,
    "adaptative": init_pheromones_adaptative,
}

add_node_strategies = {
    "base": add_node,
    "uniform": add_node_uniform
}

evaporate_pheromones_strategies = {
    "base": evaporate_pheromones,
    "exponential": evaporate_pheromones_exponential_decay,
}
```

Variante de una unica iteración

In [8]:
```python
def hormigas(
        problem: TSProblemType,
        n_hormigas: int,
        add_node_strategy: Literal["base", "uniform"] = "base",
        add_node_kwargs: Dict[str, Any] = {},
        init_pheromones_strategy: Literal["base", "adaptative"] = "base",
        init_pheromones_kwargs: Dict[str, Any] = {},
        evaporate_pheromones_strategy: Literal["base", "exponential", "adaptative"] = "base",
        evaporate_pheromones_kwargs: Dict[str, Any] = {},
) -> Tuple[SolutionType, NumericType]:
    """
    - Autor: Raúl Reyero
    ----------------------

    Algoritmo de Simulación de Colonia de Hormigas (ACO) con una unica iteración
    :param problem: datos del problema
    :param n_hormigas: Número de agentes(hormigas)
    :param add_node_strategy: ['base': aleatoria, 'uniform': Regla de Transición Aleatoria
           Proporcional]
    :param add_node_kwargs: Si la estrategia de nodos es 'uniform',
           se puede definir los valores de alfa y beta
    :param init_pheromones_strategy: ['base': aleatoria, 'adaptative': Regla de Transición Aleatoria
           Proporcional]
    :param init_pheromones_kwargs:
    :param evaporate_pheromones_strategy: ['base': aleatoria, 'exponential':
           Calcula tau_0 = 1 / (m * path_cost)]
    :param evaporate_pheromones_kwargs: Si la estrategia de nodos es 'exponential',
           se puede definir los valores de rho y min_tau
    :return: Solución y distancia recorrida
    """

    #Nodos
    n_cities = problem.dimension

    init_pheromones_callback = init_pheromones_strategies.get(init_pheromones_strategy, "base")
    add_node_callback = add_node_strategies.get(add_node_strategy, "base")
    evaporate_pheromones_callback = evaporate_pheromones_strategies.get(evaporate_pheromones_strategy, "base"

    #Inicializa las aristas con una cantidad inicial de feromonas
    T = init_pheromones_callback(problem, **init_pheromones_kwargs)

    #Se generan los agentes(hormigas) que serán estructuras de caminos desde 0
    ants = [[0] for _ in range(n_hormigas)]

    #Recorre cada agente construyendo la solución
    for h in range(n_hormigas):
        #Para cada agente se construye un camino
        for i in range(n_cities - 1):
            #Elige el siguiente nodo
            new_node = add_node_callback(problem, ants[h], T, **add_node_kwargs)
            ants[h].append(new_node)

            #Incrementa feromonas en esa arista
            T = increase_pheromone(problem, T, ants[h])
            #print("Feromonas(1)", T)

            #Evapora Feromonas
            T = evaporate_pheromones_callback(T, **evaporate_pheromones_kwargs)
            #print("Feromonas(2)", T)

            #Seleccionamos el mejor agente
        mejor_solucion = []
        mejor_distancia = float('inf')
        for h in range(n_hormigas):
            distancia_actual = distancia_total(ants[h], problem)
            if distancia_actual < mejor_distancia:
                mejor_solucion = ants[h]
                mejor_distancia = distancia_actual

        print(mejor_solucion)
        print(mejor_distancia)
        return mejor_solucion, mejor_distancia

solucion_aco_1_iter_base, distancia_actual = hormigas(tsp_problem, 1000)
```

```
[0, 13, 35, 31, 39, 34, 40, 18, 3, 15, 6, 36, 37, 17, 33, 7, 5, 11, 29, 9, 32, 21, 10, 41, 23, 24, 22, 25, 3
8, 20, 1, 28, 19, 26, 27, 2, 16, 4, 8, 30, 12, 14]
3995
```

In [9]:
```
solucion_aco_1_iter_uni_ada_exp, distancia_actual = hormigas(
    tsp_problem, 1000,
    add_node_strategy="uniform",
    init_pheromones_strategy="adaptative",
    evaporate_pheromones_strategy="exponential"
)
```

```
[0, 2, 27, 3, 18, 28, 29, 30, 21, 40, 15, 19, 5, 36, 17, 31, 1, 37, 7, 6, 25, 8, 16, 14, 26, 11, 12, 9, 23, 4
1, 10, 4, 34, 33, 20, 32, 22, 24, 13, 35, 38, 39]
3063
```

In [10]:
```
def hormigas_iterativo(
        problem: TSProblemType,
        n_hormigas: int,
        add_node_strategy: Literal["base", "uniform"] = "uniform",
        add_node_kwargs: Dict[str, Any] = {},
        init_pheromones_strategy: Literal["base", "adaptative"] = "base",
        init_pheromones_kwargs: Dict[str, Any] = {},
        evaporate_pheromones_strategy: Literal["base", "exponential", "adaptative"] = "base",
        evaporate_pheromones_kwargs: Dict[str, Any] = {},
        max_iter: int = 10,
) -> Tuple[SolutionType, NumericType]:
    """
    Implementación basada en Ant System (Dorigo et al., 1996).
    Variante: Ant-Cycle (Actualización global al final de la iteración
    propuesta en el Paper del 96').

    :param problem: datos del problema
    :param n_hormigas: Número de agentes(hormigas)
    :param add_node_strategy: ['base': aleatoria, 'uniform': Regla de Transición Aleatoria
           Proporcional]
    :param add_node_kwargs: Si la estrategia de nodos es 'uniform',
           se puede definir los valores de alfa y beta
    :param init_pheromones_strategy: ['base': aleatoria, 'adaptative': Regla de Transición Aleatoria
           Proporcional]
    :param init_pheromones_kwargs:
    :param evaporate_pheromones_strategy: ['base': aleatoria, 'exponential':
           Calcula tau_0 = 1 / (m * path_cost)]
    :param evaporate_pheromones_kwargs: Si la estrategia de nodos es 'exponential',
           se puede definir los valores de rho y min_tau
    :return: Solución y distancia recorrida
    """

    nodes = problem.cached_nodes
    n_cities = problem.dimension

    init_pheromones_callback = init_pheromones_strategies.get(init_pheromones_strategy, "base")
    add_node_callback = add_node_strategies.get(add_node_strategy, "base")
    evaporate_pheromones_callback = evaporate_pheromones_strategies.get(evaporate_pheromones_strategy, "base"
    T = init_pheromones_callback(problem, **init_pheromones_kwargs)
    best_global_solution = []
    best_global_dist = float('inf')
    for iteration in range(max_iter):
        current_gen_ants_paths = []
        for k in range(n_hormigas):
            start_node = random.choice(nodes)
            ant_path = [start_node]
            for _ in range(n_cities - 1):
                new_node = add_node_callback(problem, ant_path, T, **add_node_kwargs)
                ant_path.append(new_node)
            current_gen_ants_paths.append(ant_path)

        for path in current_gen_ants_paths:
            distance = distancia_total(path, problem)
            if distance < best_global_dist:
                best_global_dist = distance
                best_global_solution = deepcopy(path)

        T = evaporate_pheromones_callback(T, **evaporate_pheromones_kwargs)

        for path in current_gen_ants_paths:
            T = increase_pheromone(problem, T, path)


    print(best_global_solution)
    print(best_global_dist)
    return best_global_solution, best_global_dist

solucion_aco_n_iters_uni, distancia_actual = hormigas_iterativo(
    tsp_problem, 1000,
    add_node_strategy="uniform",
```

```
    max_iter=15
)
```

```
[19, 13, 5, 26, 18, 12, 11, 25, 10, 8, 9, 23, 41, 21, 40, 24, 39, 22, 38, 30, 29, 28, 2, 27, 3, 4, 6, 1, 0, 3
2, 34, 20, 33, 35, 36, 31, 17, 15, 16, 14, 37, 7]
1352
```

In [11]:
```
solucion_aco_n_iters_uni_ada_exp, distancia_actual = hormigas_iterativo(
    tsp_problem, 1000,
    add_node_strategy="uniform",
    init_pheromones_strategy="adaptative",
    evaporate_pheromones_strategy="exponential",
    max_iter=15
)
```

```
[27, 2, 3, 4, 6, 1, 0, 32, 34, 33, 20, 35, 36, 17, 31, 7, 37, 15, 16, 14, 19, 13, 5, 26, 18, 12, 11, 25, 10,
8, 9, 23, 41, 21, 40, 24, 39, 22, 38, 30, 29, 28]
1299
```

Representación en un grafo a partir de la matriz de distancias (Optimización de posiciones usando escalado multidimensional (MDS)

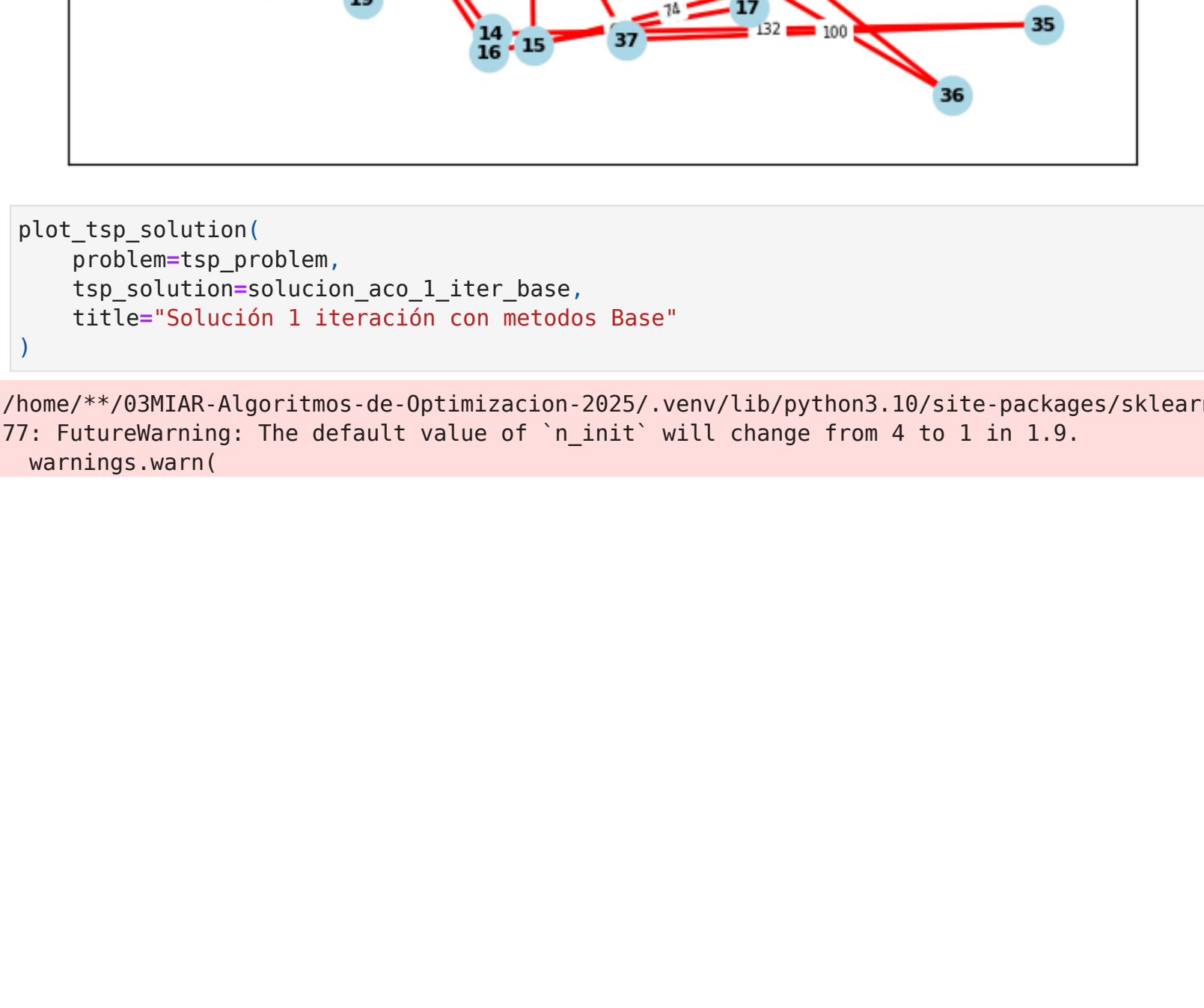
Multidimensional scaling problem(MDS): https://en.wikipedia.org/wiki/Multidimensional_scaling

In [12]:
```
import networkx as nx
import matplotlib.pyplot as plt
from sklearn.manifold import MDS  # Multidimensional Scaling o Escalado Multidimensional


def plot_tsp_solution(
        problem: TSProblemType,
        tsp_solution: SolutionType,
        title: str = "",
) -> None:
    """
    Dibuja el grafo de un TSP con las posiciones calculadas mediante MDS y muestra
    solo las aristas correspondientes a la solución del TSP.

    :param distance_matrix: np.ndarray, matriz de distancias entre nodos
    :param tsp_solution: list, lista de nodos en el orden de la solución del TSP
    """
    # Validaciones para verificar si el problema esta definido por
    # coordenadas en R2
    COORD_TYPES = {'EUC_2D', 'ATT', 'CEIL_2D', 'GEO', 'PSEUDO'}
    w_type = getattr(problem, 'edge_weight_type', None)
    has_coords = (w_type in COORD_TYPES) or (len(list(problem.node_coords)) > 0)

    if has_coords:
        pos = problem.node_coords
    else:
        distance_matrix = problem.edge_weights
        num_nodes = len(distance_matrix)
        # Usar MDS para calcular posiciones de los nodos
        mds = MDS(n_components=2, dissimilarity="precomputed", random_state=40)
        positions = mds.fit_transform(distance_matrix)
        # Convertir las posiciones en un diccionario para networkx
        pos = {i: positions[i] for i in range(num_nodes)}

    # Crear un subgrafo con las aristas del camino TSP
    TSP_G = nx.Graph()
    TSP_G.add_nodes_from(tsp_solution)
    path_len = len(tsp_solution)
    for i in range(path_len):
        u = tsp_solution[i]
        v = tsp_solution[(i + 1) % path_len]
        TSP_G.add_edge(u, v, weight=problem.get_weight(u, v))
    # Dibujar el grafo
    plt.figure(figsize=(8, 6))

    # Dibujar nodos
    nx.draw_networkx_nodes(TSP_G, pos, node_color='lightblue', node_size=250)

    # Dibujar las aristas del camino TSP
    nx.draw_networkx_edges(TSP_G, pos, edge_color='red', width=2)

    # Añadir etiquetas a los nodos y pesos de las aristas
    nx.draw_networkx_labels(TSP_G, pos, font_size=8, font_weight='bold')
    edge_labels = nx.get_edge_attributes(TSP_G, 'weight')
    nx.draw_networkx_edge_labels(TSP_G, pos, edge_labels=edge_labels,
                                 font_size=6)
```

```
    plt.title(
        f"Grafo TSP con solución específica de {problem.name}"
        f" Costo Total: {distancia_total(tsp_solution, problem)}"
        f"{f' - {title}' if title else ''}"
    )

    plt.show()
```

In [13]:
```
plot_tsp_solution(
    problem=tsp_problem,
    tsp_solution=crear_solucion(nodes),
    title="Solución Aleatoria"
)
```

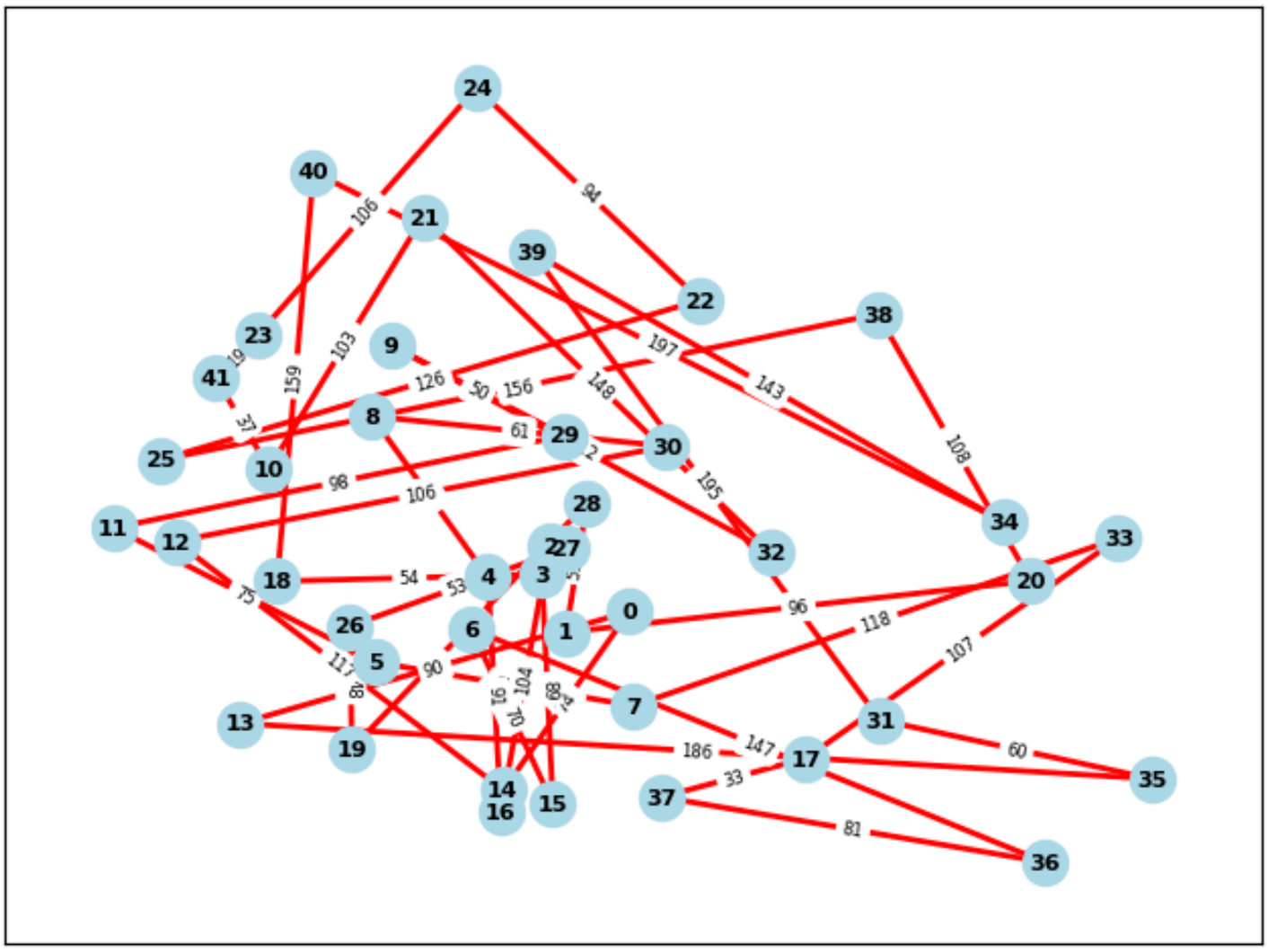
```
/home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/python3.10/site-packages/sklearn/manifold/_mds.py:677: FutureWarning: The default value of `n_init` will change from 4 to 1 in 1.9.
  warnings.warn(
```



In [14]:
```
plot_tsp_solution(
    problem=tsp_problem,
    tsp_solution=solucion_aco_1_iter_base,
    title="Solución 1 iteración con metodos Base"
)
```

```
/home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/python3.10/site-packages/sklearn/manifold/_mds.py:677: FutureWarning: The default value of `n_init` will change from 4 to 1 in 1.9.
  warnings.warn(
```

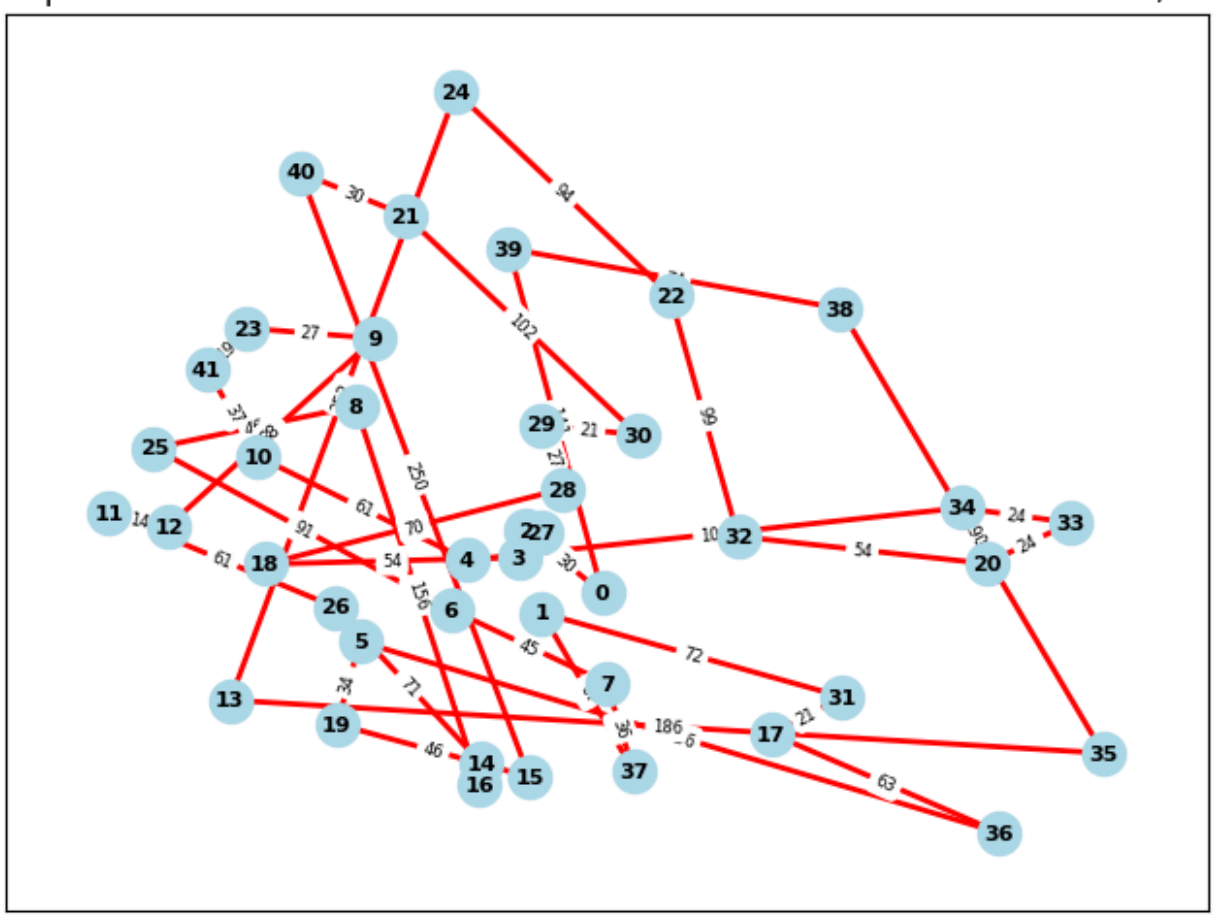
Grafo TSP con solución específica de swiss42 Costo Total: 3995 - Solución 1 iteración con metodos Base



In [15]:
```
plot_tsp_solution(
    problem=tsp_problem,
    tsp_solution=solucion_aco_1_iter_uni_ada_exp,
    title="Solución 1 iteración Uniforme, Adaptativa y Exponencial"
)
```

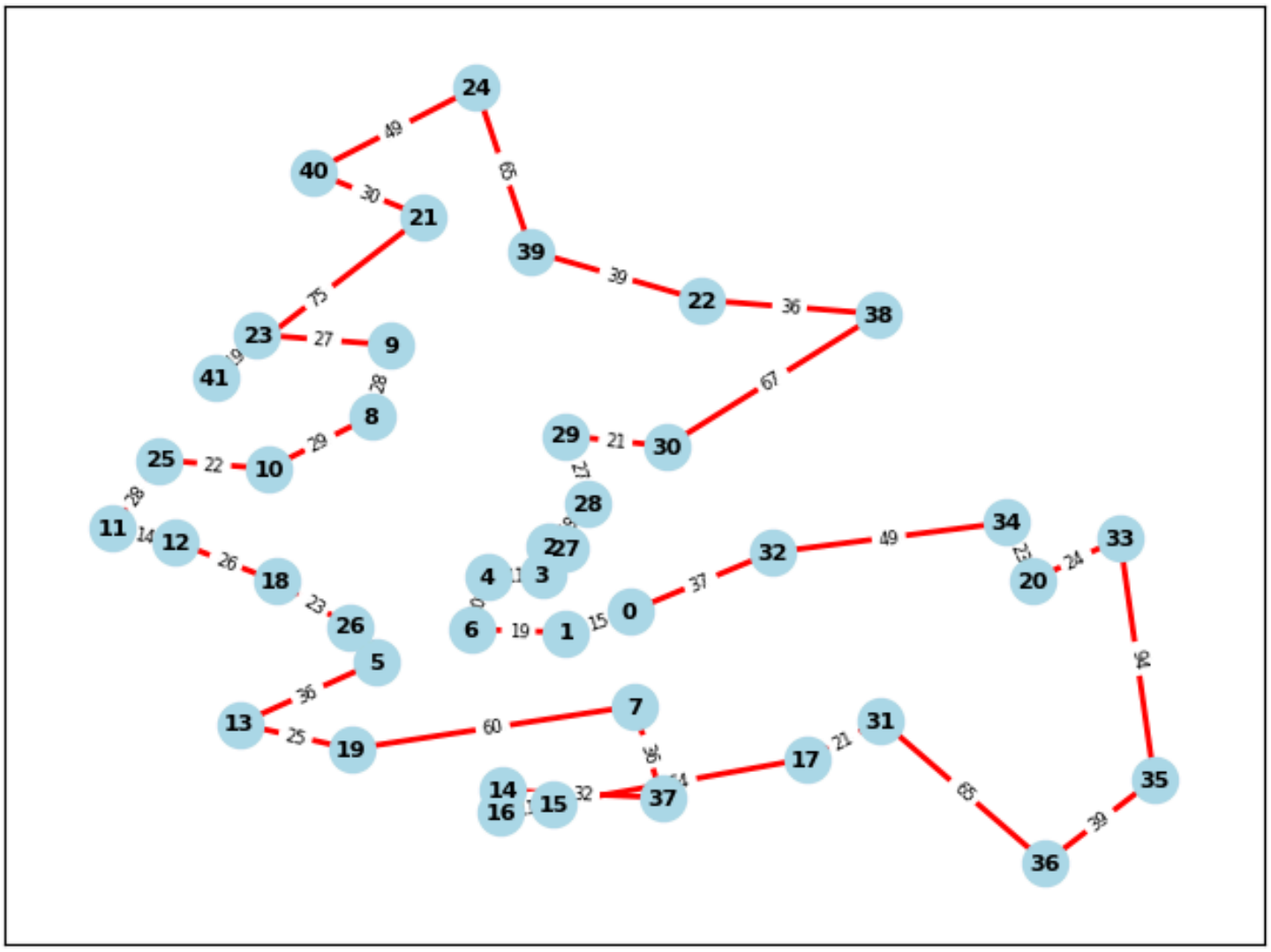
```
/home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/python3.10/site-packages/sklearn/manifold/_mds.py:677: FutureWarning: The default value of `n_init` will change from 4 to 1 in 1.9.
  warnings.warn(
```

Grafo TSP con solución específica de swiss42 Costo Total: 3063 - Solución 1 iteración Uniforme, Adaptativa y Exponencial



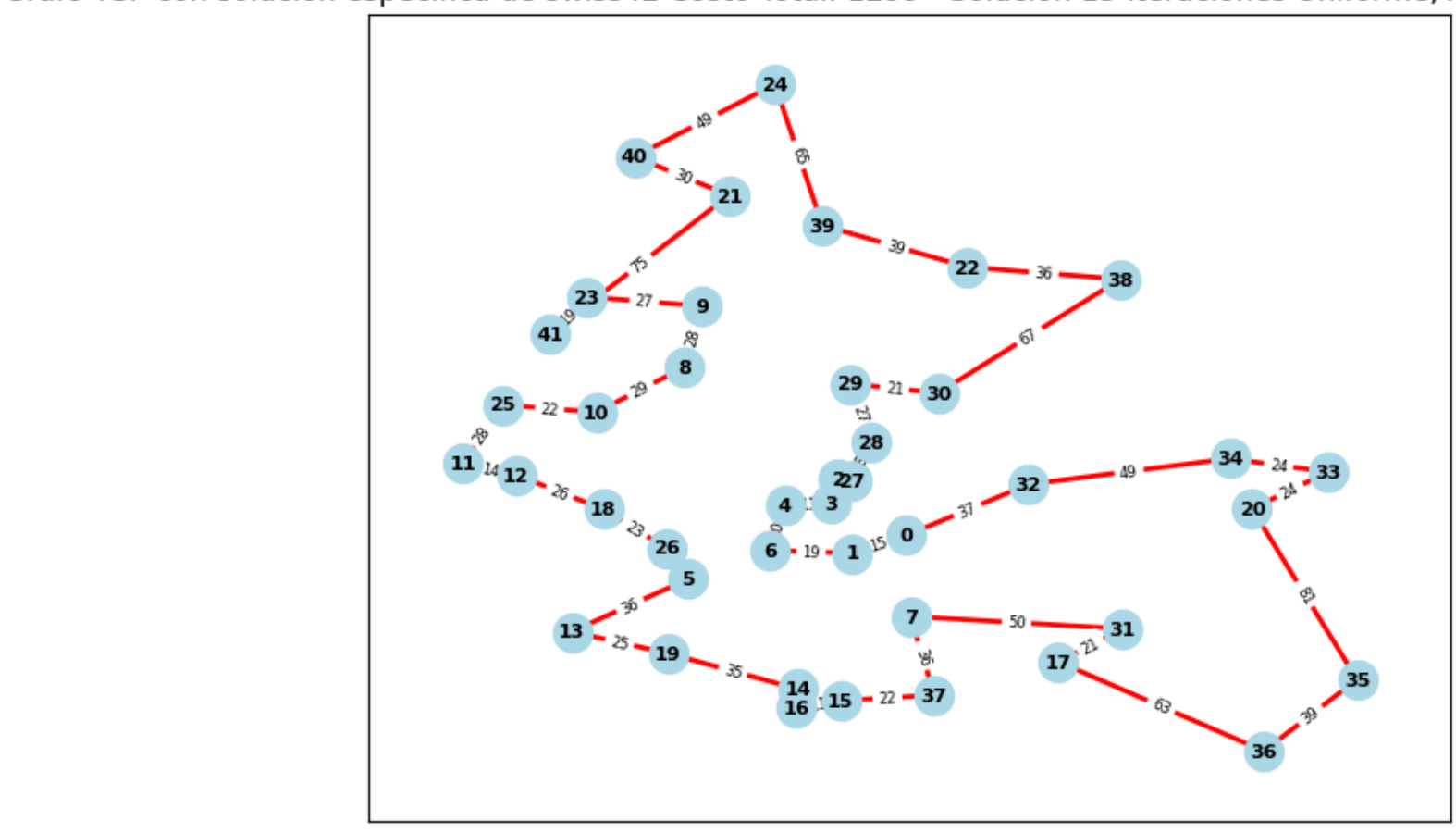
In [16]:
```
plot_tsp_solution(
    problem=tsp_problem,
    tsp_solution=solucion_aco_n_iters_uni,
    title="Solución 15 iteraciones Uniforme"
)
```

```
/home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/python3.10/site-packages/sklearn/manifold/_mds.py:677: FutureWarning: The default value of `n_init` will change from 4 to 1 in 1.9.
  warnings.warn(
```



In [17]:
```
plot_tsp_solution(
    problem=tsp_problem,
    tsp_solution=solucion_aco_n_iters_uni_ada_exp,
    title="Solución 15 iteraciones Uniforme, Adaptativa y Exponencial"
)
```

```
/home/**/03MIAR-Algoritmos-de-Optimizacion-2025/.venv/lib/python3.10/site-packages/sklearn/manifold/_mds.py:677: FutureWarning: The default value of `n_init` will change from 4 to 1 in 1.9.
  warnings.warn(
```



Bibliografía

- [1] M. Dorigo, V. Maniezzo, y A. Colorni, "Ant system: optimization by a colony of cooperating agents," IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 26, no. 1, pp. 1-13, Feb. 1996.