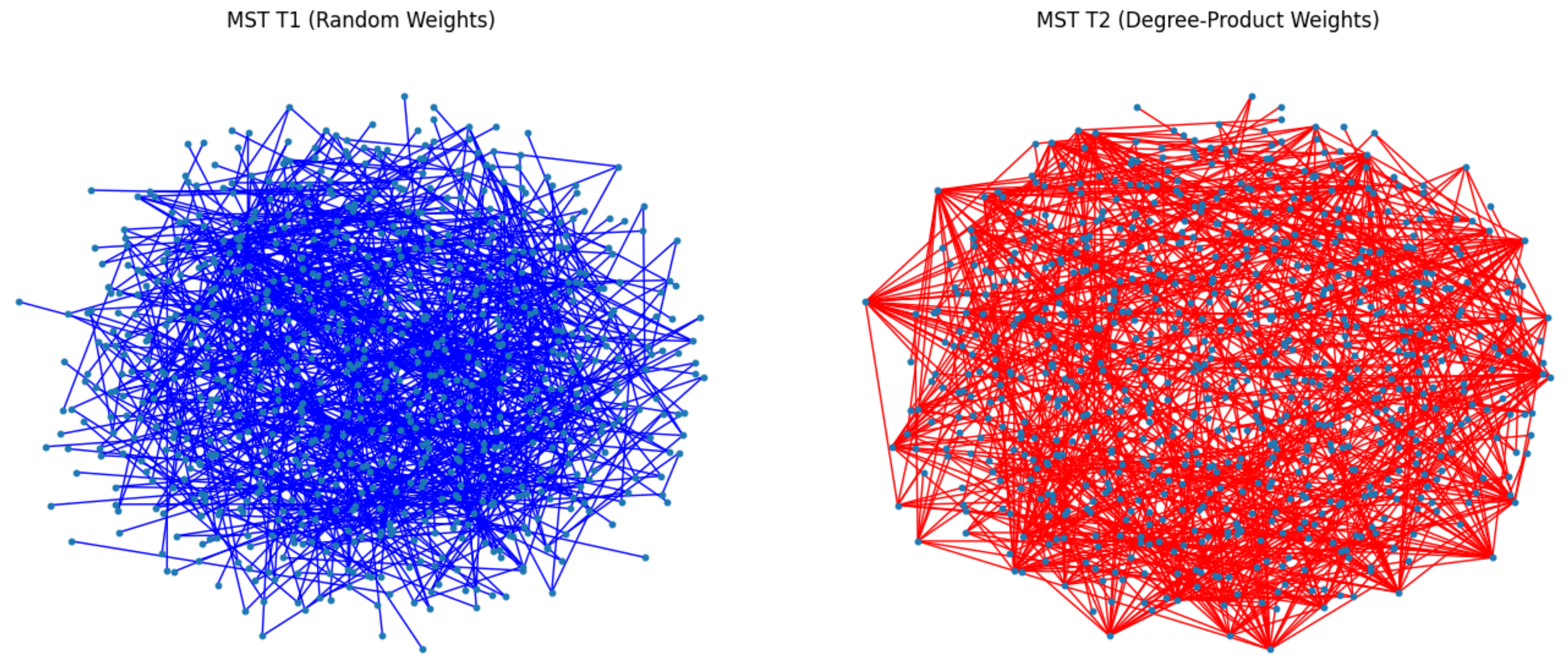
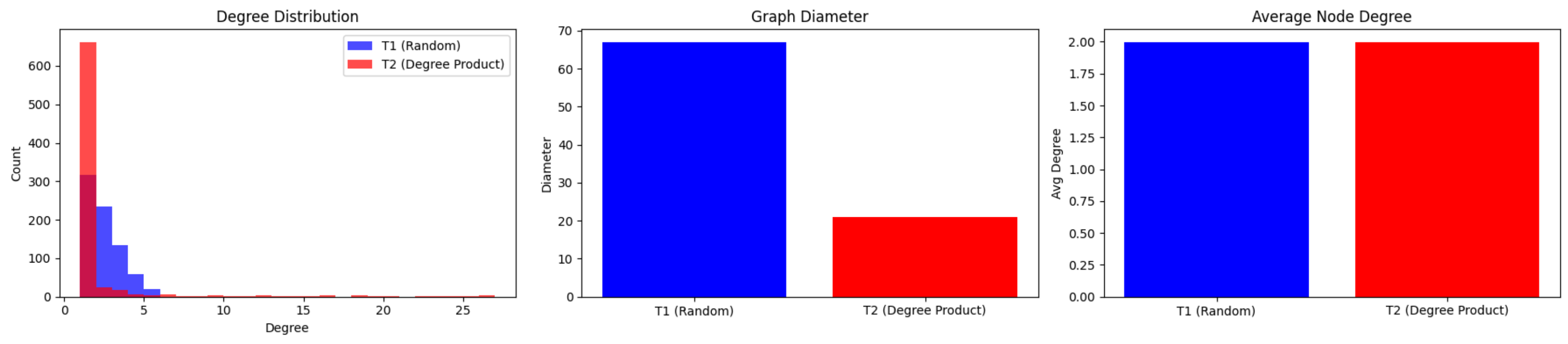
In [1]:
```python
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
```

In [2]:
```python
def Kruskal_MST(wtd_edgelist):
    def find_root(parent, i):
        if parent[i] == i:
            return i
        return find_root(parent, parent[i])

    def forest_add_edge(parent, rank, x, y):
        xroot = find_root(parent, x)
        yroot = find_root(parent, y)
        if rank[xroot] < rank[yroot]:
            parent[xroot] = yroot
        elif rank[xroot] > rank[yroot]:
            parent[yroot] = xroot
        else:
            parent[yroot] = xroot
            rank[xroot] += 1
        return parent, rank

    result = []
    i, e = 0, 0
    wtd_edgelist = sorted(wtd_edgelist, key=lambda item: item[2])
    parent = []
    rank = []
    nodes = list(set([v[0] for v in wtd_edgelist] + [v[1] for v in wtd_edgelist]))
    for node in nodes:
        parent.append(node)
        rank.append(0)
    while e < len(parent) - 1:
        u, v, w = wtd_edgelist[i]
        i = i + 1
        x = find_root(parent, u)
        y = find_root(parent, v)
        if x != y:
            e = e + 1
            result.append([u, v, w])
            parent, rank = forest_add_edge(parent, rank, x, y)

    return result
```

In [3]:
```python
# === Simulate Caltech graph with 762 nodes and 16651 edges ===
G = nx.gnm_random_graph(762, 16651, seed=42)

# === Part (i): Assign random weights and compute T1 ===
G_random = G.copy()
for u, v in G_random.edges():
    G_random[u][v]['weight'] = np.random.uniform(0, 1)
wtd_edgelist_random = [(u, v, G_random[u][v]['weight']) for u, v in G_random.edges()]
T1_edges = Kruskal_MST(wtd_edgelist_random)
T1 = nx.Graph()
T1.add_edges_from([(u, v, {'weight': w}) for u, v, w in T1_edges])

# === Part (ii): Assign degree-product weights and compute T2 ===
G_degree = G.copy()
for u, v in G_degree.edges():
    G_degree[u][v]['weight'] = G.degree[u] * G.degree[v]
wtd_edgelist_degree = [(u, v, G_degree[u][v]['weight']) for u, v in G_degree.edges()]
T2_edges = Kruskal_MST(wtd_edgelist_degree)
T2 = nx.Graph()
T2.add_edges_from([(u, v, {'weight': w}) for u, v, w in T2_edges])

# === Part (iii): Plot both MSTs ===
pos = nx.spring_layout(G, seed=42)

plt.figure(figsize=(14, 6))

# Plot T1 (Random Weights)
plt.subplot(121)
nx.draw(T1, pos, node_size=10, edge_color='blue', with_labels=False)
plt.title("MST T1 (Random Weights)")

# Plot T2 (Degree-Product Weights)
plt.subplot(122)
nx.draw(T2, pos, node_size=10, edge_color='red', with_labels=False)
plt.title("MST T2 (Degree-Product Weights)")

plt.tight_layout()
plt.show()

# === Statistical comparison ===

# Compute diameters
diameter_T1 = nx.diameter(T1)
diameter_T2 = nx.diameter(T2)

# Average degrees
avg_deg_T1 = sum(dict(T1.degree()).values()) / T1.number_of_nodes()
avg_deg_T2 = sum(dict(T2.degree()).values()) / T2.number_of_nodes()

# Degree distributions
degree_T1 = [d for n, d in T1.degree()]
degree_T2 = [d for n, d in T2.degree()]

# Plot comparison of statistics
fig, axs = plt.subplots(1, 3, figsize=(18, 4))

# Plot degree distributions
axs[0].hist(degree_T1, bins=range(1, max(degree_T1) + 1), alpha=0.7, label='T1 (Random)', color='blue')
axs[0].hist(degree_T2, bins=range(1, max(degree_T2) + 1), alpha=0.7, label='T2 (Degree Product)', color='red'
axs[0].set_title("Degree Distribution")
axs[0].set_xlabel("Degree")
axs[0].set_ylabel("Count")
axs[0].legend()

# Plot diameters
axs[1].bar(['T1 (Random)', 'T2 (Degree Product)'], [diameter_T1, diameter_T2], color=['blue', 'red'])
axs[1].set_title("Graph Diameter")
axs[1].set_ylabel("Diameter")

# Plot average degrees
axs[2].bar(['T1 (Random)', 'T2 (Degree Product)'], [avg_deg_T1, avg_deg_T2], color=['blue', 'red'])
axs[2].set_title("Average Node Degree")
axs[2].set_ylabel("Avg Degree")

plt.tight_layout()
plt.show()
```




Degree Distribution
T1 (Random)
T2 (Degree Product)
Count
Degree
Graph Diameter
Diameter
T1 (Random)
T2 (Degree Product)
Average Node Degree
Avg Degree
T1 (Random)
T2 (Degree Product)