```python
import random
import math

# Dados simulados para plataformas de transporte
plataformas = ['Uber', '99', 'Táxi']

# Parâmetros de custo para cada plataforma (Tarifa Base, Custo por Km, Custo por Minuto, Tarifa Dinâmica)
custos = {
    'Uber': {'tarifa_base': 5, 'custo_km': 1.5, 'custo_minuto': 0.5, 'tarifa_dinamica': 1.2},
    '99': {'tarifa_base': 4, 'custo_km': 1.6, 'custo_minuto': 0.4, 'tarifa_dinamica': 1.1},
    'Táxi': {'tarifa_base': 6, 'custo_km': 1.4, 'custo_minuto': 0.6, 'tarifa_dinamica': 1.3}
}

# Parâmetros da viagem (Distância em Km, Duração em Minutos, Condições Adversas)
distancia = 10  # Exemplo de distância
duracao = 20    # Exemplo de duração
condicoes_adversas = True  # Simulação de condições climáticas ou trânsito intenso

# Função para calcular o preço da viagem
def calcular_preco(plataforma, distancia, duracao, condicoes_adversas):
    tarifa = custos[plataforma]
    preco = tarifa['tarifa_base'] + (tarifa['custo_km'] * distancia) + (tarifa['custo_minuto'] * duracao)
    if condicoes_adversas:
        preco *= tarifa['tarifa_dinamica']
    return preco

# Algoritmo Hill Climb
def hill_climb(distancia, duracao, condicoes_adversas):
    melhor_preco = float('inf')
    melhor_plataforma = None
    for plataforma in plataformas:
        preco = calcular_preco(plataforma, distancia, duracao, condicoes_adversas)
        if preco < melhor_preco:
            melhor_preco = preco
            melhor_plataforma = plataforma
    return melhor_plataforma, melhor_preco

# Algoritmo Simulated Annealing
def simulated_annealing(distancia, duracao, condicoes_adversas, temperatura_inicial=100, resfriamento=0.95):
    estado_atual = random.choice(plataformas)
    melhor_estado = estado_atual
    melhor_preco = calcular_preco(estado_atual, distancia, duracao, condicoes_adversas)
    temperatura = temperatura_inicial
    while temperatura > 1:
        novo_estado = random.choice(plataformas)
        preco_atual = calcular_preco(estado_atual, distancia, duracao, condicoes_adversas)
        preco_novo = calcular_preco(novo_estado, distancia, duracao, condicoes_adversas)
        if preco_novo < preco_atual or random.uniform(0, 1) < math.exp((preco_atual - preco_novo) / temperatura):
            estado_atual = novo_estado
            if preco_novo < melhor_preco:
                melhor_preco = preco_novo
                melhor_estado = novo_estado
        temperatura *= resfriamento
    return melhor_estado, melhor_preco

# Algoritmo Genético
def algoritmo_genetico(distancia, duracao, condicoes_adversas, geracoes=50, populacao_inicial=10):
    populacao = [random.choice(plataformas) for _ in range(populacao_inicial)]
    for _ in range(geracoes):
        populacao = sorted(populacao, key=lambda x: calcular_preco(x, distancia, duracao, condicoes_adversas))
        nova_populacao = populacao[:5]
        while len(nova_populacao) < populacao_inicial:
            pai = random.choice(nova_populacao)
            mae = random.choice(nova_populacao)
            filho = random.choice([pai, mae])
            if random.random() < 0.1:  # Mutação
                filho = random.choice(plataformas)
            nova_populacao.append(filho)
        populacao = nova_populacao
    melhor_plataforma = populacao[0]
    melhor_preco = calcular_preco(melhor_plataforma, distancia, duracao, condicoes_adversas)
    return melhor_plataforma, melhor_preco

# Execução dos algoritmos
print("Hill Climb:", hill_climb(distancia, duracao, condicoes_adversas))
print("Simulated Annealing:", simulated_annealing(distancia, duracao, condicoes_adversas))
print("Algoritmo Genético:", algoritmo_genetico(distancia, duracao, condicoes_adversas))
```

```
Hill Climb: ('99', 30.800000000000004)
Simulated Annealing: ('99', 30.800000000000004)
Algoritmo Genético: ('99', 30.800000000000004)
```