Algoritmos de optimización - Seminario

Iago Monroy González:
Url: https://github.com/.../03MAIR--Algoritmos-de-Optimizacion--2019/tree/master/SEMINARIO
Problema:

> 2. Organizar los horarios de partidos de La Liga

Descripción del problema:(copiar enunciado)

Desde la La Liga de fútbol profesional se pretende organizar los horarios de los partidos de liga de cada jornada.

Se conocen algunos datos que nos deben llevar a diseñar un algoritmo que realice la asignación de los partidos a los horarios de forma que maximice la audiencia.

- Los horarios disponibles se conocen a priori y son los siguientes:
  - Viernes 20
  - Sábado 12,16,18,20
  - Domingo 12,16,18,20
  - Lunes 20
- En primer lugar se clasifican los equipos en tres categorías según el numero de seguidores(que tiene relación directa con la audiencia). Hay 3 equipos en la categoría A, 11 equipos de categoría B y 6 equipos de categoría C.
- Se conoce estadísticamente la audiencia que genera cada partido según los equipos que se enfrentan y en horario de sábado a las 20h (el mejor en todos los casos)
  - Categoría A-Categoría A -> 2 millones
  - Categoría A-Categoría B -> 1.3 millones
  - Categoría A-Categoría C -> 1 millon
  - Categoría B-Categoría B -> 0.9 millones
  - Categoría B-Categoría C-> 0.75 millones
  - Categoría C-Categoría C -> 0.47 millones
- Si el horario del partido no se realiza a las 20 horas del sábado se sabe que se reduce según los coeficientes de la siguiente tabla:
  - Viernes 20h -> 0.4
  - Sábado 12 -> 0.55
  - Sábado 16h -> 0.7
  - Sábado 18h -> 0.8
  - Sábado 20h -> 1
  - Domingo 12h -> 0.45
  - Domingo 16h -> 0.75
  - Domingo 18h -> 0.85
  - Domingo 20h -> 1
  - Lunes 20h -> 0.4
- Debemos asignar obligatoriamente siempre un partido el viernes y un partido el lunes
- Es posible la coincidencia de horarios pero en este caso la audiencia de cada partido se verá afectada y se estima que se reduce en porcentaje según la siguiente tabla dependiendo del número de coincidencias:
  - 0 -> 0%
  - 1 -> 25%
  - 2 -> 45%
  - 3 -> 60%
  - 4 -> 70%
  - 5 -> 75%
  - 6 -> 78%
  - 7 -> 80%
  - 8 -> 80%

(*) La respuesta es obligatoria

¿Cuantas posibilidades hay sin tener en cuenta las restricciones?

Sin tener en cuenta las restricciones, se tienen que distribuir 10 partidos en un total de 10 horarios disponibles. Teniendo en cuenta que el orden de las combinaciones es importante y que se permite la repetición (se puede poner más de un partido el mismo día), se puede considerar como variación con repetición, de forma que, el total de posibilidades sea de:

VR (10,10) = 10^10 = 10.000.000.000 posibilidades.

¿Cuantas posibilidades hay teniendo en cuenta todas las restricciones?

La restricción de tener que fijar partidos para lunes y viernes nos limita el espacio de soluciones considerablemente.

Al total de posibilidades anterior, habrá que restarle 2x9^10. Esto es así puesto que, aunque se fijen dos equipos, el resto puede seguir variando.

Además, habrá que sumar 8^10 combinaciones, que serán todas las restantes que no contienen ni un equipo ni el otro.

Así, el total de soluciones con restricciones queda expresado de la siguiente forma:

Total = 10^10 - 2x9^10 + 8^10 = 4.100.173.022 posibilidades.

Es decir, con una restricción tan simple como es fijar dos partidos en dos de los horarios, reducimos el espacio de soluciones en, aproximadamente, un 59% con respecto al inicial.

Modelo para el espacio de soluciones
¿Cual es la estructura de datos que mejor se adapta al problema? Argumentalo.(Es posible que hayas elegido una al principio y veas la necesidad de cambiar, arguentalo) (*)

Desde un primer momento tuve claro que utilizaría diccionarios puesto que me permite una forma mucho más organizada a la hora de estructurar los partidos, por ejemplo, junto con el número de visualizaciones que tendrá.

Sin embargo, podrían usarse listas también, aunque, en mi opinión, complicaría un poco el hecho de buscar las correspondencias entre enfrentamiento, número de visualizaciones, ponderaciones, etc.

Según el modelo para el espacio de soluciones
¿Cual es la función objetivo? (*)

La función objetivo es el número de espectadores de cada jornada.

¿Es un problema de maximización o minimización? (*)

Es un problema de maximización puesto que se desea maximizar el número de espectadores para cada jornada.

Diseña un algoritmo para resolver el problema por fuerza bruta

Calcula la complejidad del algoritmo por fuerza bruta

Diseña un algoritmo que mejore la complejidad del algortimo por fuerza bruta. Argumenta porque crees que mejora el algoritmo por fuerza bruta (*)

En mi caso, he utilizado la técnica metaheurística conocida como recocido simulado (Simulated Annealing en inglés).

Esta técnica permite mejorar el algoritmo por fuerza bruta en primer lugar por su complejidad mejor y porque, al partir de una solución inicial y tratar de mejorarla, conseguimos mejor eficiencia en el proceso de obtención de la solución.

```
### DATOS DEL PROBLEMA

from copy import deepcopy
import math
import random
from typing import Dict, List, Tuple


available_times: Dict[str, List[int]]  = {
    'V20': 0.4, 'S12': 0.55, 'S16': 0.7, 'S18': 0.8, 'S20': 1.0,
    'D12': 0.45, 'D16': 0.75, 'D18': 0.85, 'D20': 1.0, 'L20': 0.4}

team_category: Dict[str, str] = {
    "celta": "categoria B", "real madrid": "categoria A", "valencia": "categoria B", "real sociedad": "categoria A",
    "mallorca": "categoria C", "eibar": "categoria C", "athletic": "categoria B", "barcelona": "categoria A",
    "leganes": "categoria C", "osasuna": "categoria C","villareal": "categoria B", "granada": "categoria C",
    "alaves": "categoria B", "levante": "categoria B", "español": "categoria B","sevilla": "categoria B",
    "betis": "categoria B", "valladolid": "categoria C", "atletico": "categoria B", "getafe": "categoria B"}

category_audience_match: Dict[str, float] = {
```

```
    "A-A": 2.0, "A-B": 1.3,
    "A-C": 1.0, "B-B": 0.9,
    "B-C": 0.75, "C-C": 0.47}

time_correction_factor: Dict[str, Dict[int, float]] = {
    "V20": 0.4,
    "S12": 0.55,
    "S16": 0.7,
    "S18": 0.8,
    "S20": 1.0,
    "D12": 0.45,
    "D16": 0.75,
    "D18": 0.85,
    "D20": 1.0,
    "L20": 0.4,
}

coincidence_correction_factor: Dict[int, float] = {
    0: 0.0,
    1: 0.25,
    2: 0.45,
    3: 0.6,
    4: 0.7,
    5: 0.75,
    6: 0.78,
    7: 0.8,
    8: 0.8,
}

match_day: Dict[str, Dict[str, float]] = {
    "celta-real madrid": 1.3,
    "valencia-real sociedad": 1.3,
    "mallorca-eibar": 0.47,
    "athletic-barcelona": 1.3,
    "leganes-osasuna": 0.47,
    "villareal-granada": 0.75,
    "alaves-levante": 0.9,
    "español-sevilla": 0.9,
    "betis-valladolid": 0.75,
    "atletico-getafe": 0.9
}
```

```
### Funcione utilizadas en el problema
def boltzman_dist_prob(temp: float, delta: float) -> bool:
    # Checks for solution acceptation probability.
    return True if random.random() <= math.exp(-delta/temp) else False

def count_occurrences_by_daytime(matches: Dict[Tuple[str, str], Dict[str, float]]) -> Dict[str, int]:
    result = {}
    for key,_ in matches.items():
        if key[0] not in result:
            result[key[0]] = 0
        else:
            result[key[0]] += 1
    return result

def gen_neighbour_solution(matches: Dict[str, Dict[str, float]]) -> Dict[str, Dict[str, float]]:
    result = {}
    counter = 1
    aux_matches = deepcopy(matches)
    for _ in range(len(matches.keys())):
        rand_match = random.choice(list(aux_matches.keys()))
        if counter == 1:
            dt = "V20"
            result[(dt, rand_match)] = available_times[dt] * aux_matches[rand_match]
        elif counter == 2:
            dt = "L20"
            result[(dt, rand_match)] = available_times[dt] * aux_matches[rand_match]
        else:
            new_d_t = random.choice(list(available_times.keys()))
            result[(new_d_t, rand_match)] = available_times[new_d_t] * aux_matches[rand_match]
        counter += 1
        del aux_matches[rand_match]
    return list(result.keys()), get_total_audience_for_timetable_with_corrections_and_ocurrences(result)

def lower_temp(temp: float) -> float:
    return 0.99 * temp

def get_total_audience_for_timetable_with_corrections_and_ocurrences(matches: Dict[Tuple[str, str], Dict[str, float]]) -> float:
    timetable_dt_ocurrences = count_occurrences_by_daytime(matches)
    for key, value in matches.items():
        matches[key] = value * time_correction_factor[key[0]] * (1 - coincidence_correction_factor[timetable_dt_ocurrences[key[0]]])
```

```
    return sum(matches.values())

def simulated_annealing(temp: float, timetable: Dict[str, Dict[str, float]]) -> Dict[str, Dict[str, float]]:
    ref_timetable, ref_audience = gen_neighbour_solution(timetable)

    best_audience = 0
    best_timetable = ref_timetable

    while temp > 0.001:
        neighbour_timetable, neighbour_audience = gen_neighbour_solution(match_day)

        if neighbour_audience > best_audience:
            best_audience = neighbour_audience
            best_timetable = neighbour_timetable

        if neighbour_audience > ref_audience or boltzman_dist_prob(temp, abs(ref_audience - neighbour_audience)):
            ref_audience = neighbour_audience
            best_timetable = neighbour_timetable

        temp = lower_temp(temp)
    print(f"La mejor programación encontrada es: {best_timetable} con una audiencia de: {best_audience}")
    return best_timetable
```

```
simulated_annealing(10000, match_day)
```

```
La mejor programación encontrada es: [('V20', 'leganes-osasuna'), ('L20', 'mallorca-eibar'), ('D16', 'alaves-levante'), ('S20', 'at
[('V20', 'leganes-osasuna'),
 ('L20', 'mallorca-eibar'),
 ('D16', 'alaves-levante'),
 ('S20', 'atletico-getafe'),
 ('S18', 'celta-real madrid'),
 ('D18', 'betis-valladolid'),
 ('S12', 'villareal-granada'),
 ('D20', 'valencia-real sociedad'),
 ('S16', 'athletic-barcelona'),
 ('D16', 'español-sevilla')]
```

Calcula la complejidad del algoritmo (*)

La complejidad de mi algoritmo usando recocido simulado es de O(n2) siendo n el tamaño del diccionario de entrada (timetable).

Esto es así puesto que depende mayormente de la función gen_neighbour_solution que tiene dicha complejidad desmembrada en O(n) x O(n). Una de ellas por las iteraciones del bucle for y la otra la aporta la función llamada get_total_audience_for_timetable_with_corrections_and_ocurrences

Según el problema (y tenga sentido), diseña un juego de datos de entrada aleatorios

```
def generate_random_timetable() -> Dict[str, Dict[str, float]]:
    result = {}
    aux_team_category = deepcopy(team_category)
    for _ in range(int(len(team_category.keys())/2)):
        t1 = random.choice(list(aux_team_category.keys()))
        t2 = random.choice(list(aux_team_category.keys()))
        while t1 == t2:
            t2 = random.choice(list(aux_team_category.keys()))
        cat_t1 = team_category[t1].replace("categoria ", "")
        cat_t2 = team_category[t2].replace("categoria ", "")
        try:
            result[f"{t1}-{t2}"] = category_audience_match[f"{cat_t1}-{cat_t2}"]
        except:
            result[f"{t2}-{t1}"] = category_audience_match[f"{cat_t2}-{cat_t1}"]
        del aux_team_category[t1]
        del aux_team_category[t2]
    return result
```

Aplica el algoritmo al juego de datos generado

```
random_timetable = generate_random_timetable()
print(f"La jornada generada aleatoriamente es: {random_timetable}")
simulated_annealing(10000, random_timetable)
```

```
La jornada generada aleatoriamente es: {'valencia-sevilla': 0.9, 'real sociedad-mallorca': 1.0, 'celta-osasuna': 0.75, 'barcelona-v
La mejor programación encontrada es: [('V20', 'villareal-granada'), ('L20', 'mallorca-eibar'), ('D18', 'valencia-real sociedad'), (
```

```
[('V20', 'villareal-granada'),
 ('L20', 'mallorca-eibar'),
 ('D18', 'valencia-real sociedad'),
 ('S18', 'athletic-barcelona'),
 ('S20', 'leganes-osasuna'),
 ('S16', 'betis-valladolid'),
 ('D16', 'alaves-levante'),
 ('D20', 'español-sevilla'),
 ('V20', 'atletico-getafe'),
 ('S20', 'celta-real madrid')]
```

Enumera las referencias que has utilizado(si ha sido necesario) para llevar a cabo el trabajo

Para llevar a cabo este trabajo he utilizado única y exclusivamente los recursos disponibles de la asignatura. En concreto, las diapositivas sobre metaheurísticas para ver el algoritmo del recocido simulado.

Describe brevemente las lineas de como crees que es posible avanzar en el estudio del problema. Ten en cuenta incluso posibles variaciones del problema y/o variaciones al alza del tamaño

La verdad es que me ha quedado “lástima” de no haber tenido más tiempo para tratar de aplicar un algoritmo genético. ¿Por qué un algoritmo genético? Porque habiendo visto la teoría sobre ellos me parecen muy interesantes además de que me gustaría comprobar como de bien se pueden aplicar a todo tipo de problemas teniendo en cuenta su flexibilidad. Por último, también me gustaría comprobar su eficiencia a la hora de resolver este problema en concreto.

✓ 0 s completado a las 18:59