UNIVERSIDAD
DE GRANADA

TSCAO

MÁSTER CIENCIA DE DATOS E INGENIERÍA DE COMPUTADORES

Metaheurísticas

Trabajo final

Autor
Ignacio Vellido Expósito
ignaciove@correo.ugr.es

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

Curso 2020-2021

1. Maximum Diversity Problem (MD)

1.1. Búsqueda bibliográfica

- Lopez-Pires, Fabio & Vera, Katherine & Baran, Benjamin & Sandoya, Fernando. (2017). Multi-Objective Maximum Diversity Problem. 10.1109/CLEI.2017.8226423.
- Marti, Rafael & Gallego, Micael & Duarte, Abraham. (2010). A branch and bound algorithm for the maximum diversity problem. European Journal of Operational Research. 200. 36-44. 10.1016/j.ejor.2008.12.023.
- Aringhieri, Roberto & Cordone, Roberto. (2008). Tabu Search versus GRASP for the maximum diversity problem. 4OR. 6. 10.1007/s10288-007-0033-9.
- Parreño, Francisco & ÁLvarez-Valdés, Ramón & Marti, Rafael. (2020). Measuring Diversity. A review and an empirical analysis. European Journal of Operational Research. 289. 10.1016/j.ejor.2020.07.053.
- Marti, Rafael & Martínez-Gavara, Anna & Sánchez-Oro, Jesús. (2021). The capacitated dispersion problem: an optimization model and a memetic algorithm. Memetic Computing. 13. 10.1007/s12293-020-00318-1.
- Marti, Rafael & Gallego, Micael & Duarte, Abraham & G. Pardo, Eduardo. (2013). Heuristics and metaheuristics for the maximum diversity problem. Journal of Heuristics - HEURISTICS. 19. 1-25. 10.1007/s10732-011-9172-4.
- Silva, Geiza & Ochi, Luiz & Martins, Simone. (2004). Experimental Comparison of Greedy Randomized Adaptive Search Procedures for the Maximum Diversity Problem. Lecture Notes on Computer Science. 3059. 498-512. 10.1007/978-3-540-24838-5_37.
- Zhou, Yangming & Hao, Jin-Kao & Duval, Beatrice. (2017). Opposition-Based Memetic Search for the Maximum Diversity Problem. IEEE Transactions on Evolutionary Computation. 21. 731-745. 10.1109/TEVC.2017.2674800.
- Gallego, Micael & Duarte, Abraham & Laguna, Manuel & Marti, Rafael. (2009). Hybrid heuristics for the maximum diversity problem. Computational Optimization and Applications. 44. 411-426. 10.1007/s10589-007-9161-6.
- Silva, Geiza & Andrade, Marcos & Ochi, Luiz & Martins, Simone & Plastino, Alexandre. (2007). New heuristics for the maximum diversity problem. J. Heuristics. 13. 315-336. 10.1007/s10732-007-9010-x.
- Santos, L. & Ribeiro, Marcos & Plastino, Alexandre & Martins, Simone. (2005). A Hybrid GRASP with Data Mining for the Maximum Diversity Problem. Lecture Notes in Computer Science. 3636. 116-127. 10.1007/11546245_11.
- Zhou, Yalan & Yin, Jian & Zhang, Yunong. (2009). Competitive Hopfield Network Combined With Estimation of Distribution for Maximum Diversity Problems. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on. 39. 1048 - 1066. 10.1109/TSMCB.2008.2010220.
- Andrade, Marcos & Andrade, Paulo & Martins, Simone & Plastino, Alexandre. (2005). GRASP with Path-Relinking for the Maximum Diversity Problem. Lecture Notes in Computer Science. 3503. 558-569. 10.1007/11427186_48.

- Lozano, Manuel & Molina, Daniel & GarcIÂ´a-MartIÂ´nez, C.. (2011). Iterated greedy for the maximum diversity problem. European Journal of Operational Research. 214. 31-38. 10.1016/j.ejor.2011.04.018.

1.2. Pseudocódigos

1.2.1. Greedy

Un algoritmo greedy se define de la siguiente forma:

Algorithm 1: Pseudocódigo algoritmo greedy

Input: datos: Conjunto de datos
sol = solución actual vacía ;
repeat
  sol += elegirCandidatoGreedy(sol, datos) ;
until *hasta que sol sea un solución*;
return *sol*

Pudiendo ser **elegirCandidatoGreedy**:

Algorithm 2: elegirCandidatoGreedy

Input: sol, datos
return *candidato no presente en sol con mayor distancia a los elementos en sol*

1.2.2. Semi-Greedy

Un algoritmo semi-greedy se define de la siguiente forma:

Algorithm 3: Pseudocódigo algoritmo semi-greedy

Input: datos: Conjunto de datos
sol = solución actual vacía ;
repeat
  candidatos = elegirMejoresCandidatosGreedy(sol, datos) ;
  sol += random(candidatos) ;
until *hasta que sol sea un solución*;
return *sol*

Pudiendo ser **elegirMejoresCandidatosGreedy**:

Algorithm 4: elegirMejoresCandidatosGreedy

Input: sol, datos
return *N candidatos no presentes en sol ordenados por mayor distancia a sol*

1.2.3. Iterated-Greedy

Un algoritmo de iterated-greedy se define de la siguiente forma:

Algorithm 5: Pseudocódigo algoritmo iterated-greedy

```
Input: datos: Conjunto de datos
sol = solución actual vacía ;
repeat
    xd = destrucción(sol) ;
    xc = construcción(xd, datos) ;
    sol = aceptar(sol, xc) ;
until hasta que sol cumpla los criterios de parada;
return sol
```

Pudiendo ser **destrucción**:

Algorithm 6: destrucción

```
Input: sol
sol = quitar N elementos aleatorios de la solución sol ;
return sol
```

Pudiendo ser **construcción**:

Algorithm 7: construcción

```
Input: sol, datos
return candidato no presente en sol con mayor distancia a los elementos en sol
```

Pudiendo ser **aceptar**:

Algorithm 8: aceptar

```
Input: sol, temp
best = sol ;
if diversidad(temp) > diversidad(sol) then
    best = temp ;
return best
```

1.3. Búsqueda local

Algorithm 9: Pseudocódigo algoritmo de búsqueda local

```
Input: datos: Conjunto de datos
sol = solución aleatoria válida ;
candidatos = vecindario(sol) ;
repeat
    sol = best(candidatos) ;
    candidatos = vecindario(sol) ;
until hasta que candidatos esté vacía;
return sol
```

Pudiendo ser un posible operador de vecindario:

Algorithm 10: Operador de vecindario

```
Input: sol, datos
return sustituir elemento i de sol por elemento j
```

1.4. Algoritmo genético

Una posible representación podría ser un vector binario $x = (x_0, ..., x_n)$ indicando si un elemento x_i está presente o no en el subconjunto solución.

De esta forma tendríamos como operadores de cruce y mutación:

Algorithm 11: Operador de cruce

```
Input: sol1, sol2
sol = solución vacía ;
first = seleccionar n/2 elementos incluídos en sol1 ;
second = seleccionar n/2 elementos incluídos en sol2 ;
sol = first + second ;
/* Reparar                                              */
sol = añadir o quitar elementos aleatoriamente hasta que sol sea válida ;
return sol
```

Algorithm 12: Operador de mutación

```
Input: sol
sol = quitar n elementos de sol aleatoriamente y añadir n aleatoriamente ;
return sol
```

Y una inicialización correspondería a seleccionar m elementos de x de forma aleatoria.

2. Multidimensional two-way number partitioning problem (M2NP)

2.1. Búsqueda bibliográfica

- Kojić, Jelena. (2010). Integer linear programming model for multidimensional two-way number partitioning problem. Computers & Mathematics with Applications. 60. 2302-2308. 10.1016/j.camwa.2010.08.024.

- Alexandre Frias Faria, Sérgio Ricardo de Souza, Elisangela Martins de Sá, A mixed-integer linear programming model to solve the Multidimensional Multi-Way Number Partitioning Problem, Computers & Operations Research, Volume 127, 2021, 105133, ISSN 0305-0548.

- Santucci, Valentino & Baioletti, Marco & Di Bari, Gabriele & Milani, Alfredo. (2019). A Binary Algebraic Differential Evolution for the MultiDimensional Two-Way Number Partitioning Problem. 10.1007/978-3-030-16711-0_2.

- Hacibeyoglu, Mehmet & Alaykiran, Kemal & ACILAR, A. Merve & Tongur, Vahit & Ülker, Erkan. (2018). A Comparative Analysis of Metaheuristic Approaches for Multidimensional Two-Way Number Partitioning Problem. Arabian Journal for Science and Engineering. 43. 10.1007/s13369-018-3155-9.

- Jozef Kratica, Jelena Kojić, Aleksandar Savić,Two metaheuristic approaches for solving multidimensional two-way number partitioning problem,Computers & Operations Research,Volume 46,2014,Pages 59-68,ISSN 0305-0548,

- Pop, Petrica & Matei, Oliviu. (2013). A Genetic Algorithm Approach for the Multidimensional Two-Way Number Partitioning Problem. 7997. 81-86. 10.1007/978-3-642-44973-4_10.

- Petrică C. Pop, Oliviu Matei,A memetic algorithm approach for solving the multidimensional multi-way number partitioning problem,Applied Mathematical Modelling,Volume 37, Issue 22,2013,Pages 9191-9202,ISSN 0307-904X,

- Vera, J. & Macías, Rodrigo & Heiser, Willem. (2009). A Latent Class Multidimensional Scaling Model for Two-Way One-Mode Continuous Rating Dissimilarity Data. Psychometrika. 74. 297-315. 10.1007/s11336-008-9104-x.

2.2. Pseudocódigos

2.2.1. Greedy

Siendo t el valor de la función objetivo a minimizar, un posible algoritmo greedy se define de la siguiente forma:

Algorithm 13: Pseudocódigo algoritmo greedy

```
Input: datos: Conjunto de datos
sol = solución actual vacía ;
i = 0 ;
repeat
    sol = elegirCandidatoGreedy(sol, datos, i) ;
    i++ ;
until hasta que sol sea un solución;
return sol
```

Pudiendo ser **elegirCandidatoGreedy**:

Algorithm 14: elegirCandidatoGreedy

```
Input: sol, datos, i
sol = añadir datos_i al subconjunto de sol que menos incremente t ;
return sol
```

2.2.2. Semi-Greedy

Un algoritmo semi-greedy se define de la siguiente forma:

Algorithm 15: Pseudocódigo algoritmo semi-greedy

```
Input: datos: Conjunto de datos
sol = solución actual vacía ;
repeat
    candidatos = elegirMejoresCandidatosGreedy(sol, datos) ;
    sol += random(candidatos) ;
until hasta que sol sea un solución;
return sol
```

Pudiendo ser **elegirMejoresCandidatosGreedy**:

Algorithm 16: elegirMejoresCandidatosGreedy

```
Input: sol, datos
return N candidatos no presentes en sol ordenados por menor incremento de t en
  cualquier subconjunto
```

2.2.3. Iterated-Greedy

Un algoritmo de iterated-greedy se define de la siguiente forma:

Algorithm 17: Pseudocódigo algoritmo iterated-greedy

```
Input: datos: Conjunto de datos
sol = solución actual vacía ;
repeat
    xd = destrucción(sol) ;
    xc = construcción(xd, datos) ;
    sol = aceptar(sol, xc) ;
until hasta que sol cumpla los criterios de parada;
return sol
```

Pudiendo ser **destrucción**:

Algorithm 18: destrucción

```
Input: sol
sol = quitar elemento aleatorio de un subconjunto de sol ;
return sol
```

Pudiendo ser **construcción**:

Algorithm 19: construcción

```
Input: sol, datos
return candidato no presente en sol que menos incremente t
```

Pudiendo ser **aceptar**:

Algorithm 20: aceptar

Input: sol, temp
best = sol ;
if $t_{temp} < t_{sol}$ **then**
    best = temp ;
return *best*

2.3. Búsqueda local

Algorithm 21: Pseudocódigo algoritmo de búsqueda local

```
Input: datos: Conjunto de datos
sol = solución aleatoria válida ;
candidatos = vecindario(sol) ;
repeat
    sol = best(candidatos) ;
    candidatos = vecindario(sol) ;
until hasta que candidatos esté vacía;
return sol
```

Pudiendo ser un posible operador de vecindario:

Algorithm 22: Operador de vecindario

```
Input: sol, datos
return sol cambiando elemento i de subconjunto
```

2.4. Algoritmo genético

Una posible representación podría ser un vector binario $x = (x_0, ..., x_n)$ indicando a qué subconjunto un elemento x_i pertenece.

De esta forma tendríamos como operadores de cruce y mutación:

Algorithm 23: Operador de cruce

```
Input: sol1, sol2
sol = solución vacía ;
first = asignación en subconjuntos de n/2 elementos de sol1 ;
second = asignación en subconjuntos de n/2 elementos de sol2 ;
sol = first + second ;
/* Reparar                                                   */
sol = incluir o quitar elementos aleatoriamente de los subconjuntos hasta que sol
  sea válida ;
return sol
```

Algorithm 24: Operador de mutación

```
Input: sol
sol = cambiar n elementos de subconjunto aleatoriamente ;
return sol
```

Y una inicialización asignar un subconjunto a cada elemento de forma aleatoria.