Computer Science

Max diversity informe

Adrián Fleitas de la Rosa

alu
alu0101024363

Contents

1 Estructura de datos

Se ha obtado por un vector de vectores de double ya que los vectores dados pueden tener mas de dos dimensiones. Para la implementación de los diferentes métodos de resolución del maximun diversity se optó por el patrón de diseño estrategia y por guardar la solución encapsulada en una clase. Por tanto nos queda un esquema de clase siguiente: La clase maxdiversity utiliza usa la clase solution para devolver la solución dada por uno de los algoritmos usados(usando sus respectivas clases derivadas). La clase abstracta algoritmo nos permite utilizar el patrón estrategia de ella derivan tanto la clase greedy como la grasp y la branch. Cada una de ellas implementa el algoritmo que define su nombre. La clase solución representa una posible resolución al problema y aporta métodos de utilidad como la distancia entre x e y vectores, la distancia total, el tiempo de cpu, etc.

2 Algoritmos utilizados

Se han implementado tres principales algoritmos de resolución de maximum diversity: Greedy, Grasp y Branch and Bound.

2.1 Greedy

En este algoritmo simplemente vamos a escoger el vector mas alejado del centroide paso a paso siguiendo como su propio nombre indica una estrategia greedy. Primero recogemos la información del problema (maxdiversity) , establecemos el número de vectores y su dimensión asi como recorremos el valor m como parámetro. Consideramos m como el tamaño de la solución. Luego mientras el tamaño de la solución que estamos generando no sea superior a la m dada escogemos el elemento mas lejano. Finalmente usando nuestra clase Solution devolvemos el resultado hayado.

```
2
3   Solution Greedy::solve(Maxdiversity max, int m, int rlc) {
4     Solution initial(max.getPointsCopy(), max.getDimension());
5     Solution finalSolution;
6     finalSolution.setK(max.getDimension());
7     auto sc = initial.centroid();
8     int i = 0;
9     auto t_start = std::chrono::high_resolution_clock::now();
10    while (i < m) {
11      auto se = initial.getFarElementRemoving(sc);
12      finalSolution.add(se);
13      sc = finalSolution.centroid();
14      i++;
15    }
16    auto t_end = std::chrono::high_resolution_clock::now();
17    auto timeCost = std::chrono::duration<double, std::milli>(t_end-t_start).count();
18    finalSolution.setTimeCost(timeCost);
19    return finalSolution;
20  }
21
```

2.2 Grasp

Este algoritmo se compone de dos partes: una primera parte de construcción y una segunda de búsqueda local hasta encontrar un óptimo local. En esta primera parte usando un acercamiento greedy pero con un componente aleatorio lrc (que será posteriormente explicado) generamos una primera solución factible que no óptima. Y a partir de esta usamos un procedimiento de búsqueda local (cambio de vector de la solución por otro no de la solución actual) para encontrar el óptimo local es decir la mejor solución tras hacer un solo intercambio. RLC: este algoritmo dependiendo de un límite de vectores dado, escoge los n mejores vectores siguiendo el criterio greedy para el siguiente paso del grasp. El grasp de esta lista de vectores escoge aleatoriamente uno de ellos para añadir a la ruta de ahí el componente aleatorio. Finalmente como el resto de algoritmos grasp devuelve la mejor solución factible que ha encontrado.

```cpp
Solution Grasp::solve(Maxdiversity max, int m, int rlc) {
  auto t_start = std::chrono::high_resolution_clock::now();
  Solution finalSolution = ConstructGrasp(max, m, rlc);
  int i = 0;
  while (i < 20) {
    Solution noLocal = ConstructGrasp(max, m, rlc);
    Solution sol = localSearch(max,noLocal);
    //cout << "Sol no local: " << noLocal.distanceTotal() << endl;
    if (sol.distanceTotal() > finalSolution.distanceTotal()) {
      finalSolution = sol;
    }
    i++;
  }
  auto t_end = std::chrono::high_resolution_clock::now();
  auto timeCost = std::chrono::duration<double, std::milli>(t_end-t_start).count();
  //noLocal.print();
  finalSolution.setTimeCost(timeCost);
  return finalSolution;
}
```

```cpp
Solution Grasp::ConstructGrasp(Maxdiversity max, int m, int rlc) {
  Solution initial(max.getPointsCopy(), max.getDimension());
  Solution finalSolution;
  finalSolution.setK(max.getDimension());
  auto sc = initial.centroid();
  int i = 0;
  while (i < m) {
    vector<vector<double>> elements;
    Solution temp = initial;
    for (int i = 0; i < rlc; i++) {
      elements.push_back(temp.getFarElementRemoving(sc));
    }
    int random = rand() % elements.size();
    auto se = elements[random];
    //cout << random << endl;
    initial.remove(se);
    finalSolution.add(se);
    sc = finalSolution.centroid();
    i++;
  }
  return finalSolution;
}
```

2.3 Branch and Bound

A diferencia del resto de algoritmos propuesto Branch and Bound es un algoritmo exacto que a través del uso de cotas superiores e inferiores consigue no explorar todas las posibles soluciones(poda). Se establece como cota inferior inicial la dada por un algoritmo greedy y como superior la multiplicación por la mayor distancia.

2.4 Tablas

Greedy

```
AdrianKousel@DESKTOP-LB4LH48 MINGW64 /e/VS code proyect/
$ ./max
Problema           n   K   m        z       CPU
max_div_15_2.txt  15   2   2     11.8592   2.998
max_div_15_2.txt  15   2   3     25.7262   2.997
max_div_15_2.txt  15   2   4     48.4139   7.991
max_div_15_2.txt  15   2   5     73.5619    5.02

max_div_15_3.txt  15   3   2     13.2732   2.996
max_div_15_3.txt  15   3   3     30.3241   2.998
max_div_15_3.txt  15   3   4     59.7638   4.995
max_div_15_3.txt  15   3   5     94.7487   4.649

max_div_20_2.txt  20   2   2     8.51033   2.998
max_div_20_2.txt  20   2   3     21.9961   3.996
max_div_20_2.txt  20   2   4     39.5682   5.018
max_div_20_2.txt  20   2   5     62.1989   5.994

max_div_20_3.txt  20   3   2     11.8003   2.996
max_div_20_3.txt  20   3   3     30.8727   3.996
max_div_20_3.txt  20   3   4     56.6903   4.995
max_div_20_3.txt  20   3   5     92.8297   6.993

max_div_30_2.txt  30   2   2     11.6571   3.991
max_div_30_2.txt  30   2   3     28.9443   4.996
max_div_30_2.txt  30   2   4     52.7712   8.991
max_div_30_2.txt  30   2   5     80.9102  10.988

max_div_30_3.txt  30   3   2     13.0737   3.996
max_div_30_3.txt  30   3   3     33.8423   5.993
max_div_30_3.txt  30   3   4      63.702   7.993
max_div_30_3.txt  30   3   5      99.592   9.989
```

Grasp

Problema	n	K	m	LRC	z	CPU
max_div_15_2.txt	15	2	2	1	11.8592	1.998
max_div_15_2.txt	15	2	2	2	11.8592	1.998
max_div_15_2.txt	15	2	2	3	11.8592	1.999
max_div_15_2.txt	15	2	3	1	25.7262	2.998
max_div_15_2.txt	15	2	3	2	25.7262	2.996
max_div_15_2.txt	15	2	3	3	25.7262	2.997
max_div_15_2.txt	15	2	4	1	48.4139	3.996
max_div_15_2.txt	15	2	4	2	49.0107	4.988
max_div_15_2.txt	15	2	4	3	49.5462	3.996
max_div_15_2.txt	15	2	5	1	73.5619	4.995
max_div_15_2.txt	15	2	5	2	77.787	5.994
max_div_15_2.txt	15	2	5	3	75.6086	7.992
max_div_15_3.txt	15	3	2	1	13.2732	1.998
max_div_15_3.txt	15	3	2	2	13.2732	2.997
max_div_15_3.txt	15	3	2	3	13.2732	2.997
max_div_15_3.txt	15	3	3	1	30.3241	2.996
max_div_15_3.txt	15	3	3	2	30.3241	2.997
max_div_15_3.txt	15	3	3	3	30.3241	3.997
max_div_15_3.txt	15	3	4	1	59.7638	3.996
max_div_15_3.txt	15	3	4	2	59.7638	4.995
max_div_15_3.txt	15	3	4	3	59.7638	4.996
max_div_15_3.txt	15	3	5	1	94.7487	4.995
max_div_15_3.txt	15	3	5	2	94.7487	5.994
max_div_15_3.txt	15	3	5	3	96.0858	5.993

```
max_div_20_2.txt  20   2   2   1 8.51033   1.998
max_div_20_2.txt  20   2   2   2 8.51033   1.998
max_div_20_2.txt  20   2   2   3 8.51033   2.998
max_div_20_2.txt  20   2   3   1 21.9961   3.995
max_div_20_2.txt  20   2   3   2 21.1324   4.993
max_div_20_2.txt  20   2   3   3 21.9961   4.995
max_div_20_2.txt  20   2   4   1 39.5682   4.995
max_div_20_2.txt  20   2   4   2 39.8292   5.994
max_div_20_2.txt  20   2   4   3 39.8292   7.523
max_div_20_2.txt  20   2   5   1 62.1989   6.992
max_div_20_2.txt  20   2   5   2 62.8729   6.992
max_div_20_2.txt  20   2   5   3 63.6517   6.993

max_div_20_3.txt  20   3   2   1 11.8003   2.997
max_div_20_3.txt  20   3   2   2 11.8003   2.997
max_div_20_3.txt  20   3   2   3 11.8003   2.997
max_div_20_3.txt  20   3   3   1 30.8727   3.997
max_div_20_3.txt  20   3   3   2 30.8727   3.996
max_div_20_3.txt  20   3   3   3 29.4688   4.995
max_div_20_3.txt  20   3   4   1 56.6903   5.994
max_div_20_3.txt  20   3   4   2 56.6903   5.994
max_div_20_3.txt  20   3   4   3 56.6903   9.498
max_div_20_3.txt  20   3   5   1 92.8297   6.993
max_div_20_3.txt  20   3   5   2 92.8297   7.997
max_div_20_3.txt  20   3   5   3 92.8297   8.991

max_div_30_2.txt  30   2   2   1 11.6571   2.997
max_div_30_2.txt  30   2   2   2 11.6571   2.996
max_div_30_2.txt  30   2   2   3 11.6571   4.995
max_div_30_2.txt  30   2   3   1 28.9443   5.994
max_div_30_2.txt  30   2   3   2 28.9443   5.994
max_div_30_2.txt  30   2   3   3 28.9443   5.994
max_div_30_2.txt  30   2   4   1 52.7712   7.544
max_div_30_2.txt  30   2   4   2 52.7712   8.974
max_div_30_2.txt  30   2   4   3 52.7712   8.991
max_div_30_2.txt  30   2   5   1 80.9102   8.991
max_div_30_2.txt  30   2   5   2 80.9102  10.989
max_div_30_2.txt  30   2   5   3 80.9102  12.987
```

```
max_div_30_3.txt  30   3   2   1 13.0737   3.995
max_div_30_3.txt  30   3   2   2 13.0737   4.996
max_div_30_3.txt  30   3   2   3 13.0737   4.995
max_div_30_3.txt  30   3   3   1 33.8423   5.994
max_div_30_3.txt  30   3   3   2 33.5049   6.993
max_div_30_3.txt  30   3   3   3 34.2905   7.993
max_div_30_3.txt  30   3   4   1  63.702   7.992
max_div_30_3.txt  30   3   4   2  63.702   8.567
max_div_30_3.txt  30   3   4   3  63.702    9.99
max_div_30_3.txt  30   3   5   1  99.592  10.988
max_div_30_3.txt  30   3   5   2  99.592  12.986
max_div_30_3.txt  30   3   5   3 98.7406  13.984
```

Link del latex: https://www.overleaf.com/read/vwtwzrjbkhpy