Ramificación y Acotación

DANIEL HERNÁNDEZ DE LEÓN

May 2022

1 Introduction

Se pide implementar dos algoritmos exactos y dos algoritmos aproximados. Los exactos son Voráz y Ramificación y Poda; y los aproximados son GRASP y Tabu.

Todos ellos para resolver un problema de Máxima Dispersión en la que tenemos N vectores de M componentes y tenemos que encontrar los O vectores que maximizen la dispersión entre ellos.

2 Voraz

2.1 Pseudocódigo

function GREEDY(vectors, lenSol)
    $solution \leftarrow empty$
    $centerVector \leftarrow center(vectors)$
    $solution \leftarrow farthestFrom(centerVector)$
    while len(solution) != lenSol **do**
        $centerVector \leftarrow center(solution)$
        $solution \leftarrow farthestFrom(centerVector)$
    end while
    return solution
end function

2.2 Explicación

Este algoritmo es exacto y se basa en elegir siempre el mejor candidato posible, en este caso, el mejor candidatos es el más alejado al centro actual de los vectores, y cada vez que metamos un vector nuevo calculamos el centro a partir de estos. Es bastante rápido y nos genera soluciones bastante buenas, por el contrario no es un algoritmo que genere soluciones óptimas.

2.3 Resultados

N Vect	Dim Vect	LenSol	Disp	Time
15	2	2	11.859216	0.0013209
15	2	3	25.7262	8.3E-06
15	2	4	48.41393	6.4E-06
15	2	5	73.5619	1.01E-05
15	3	2	13.27324	5.2E-06
15	3	3	30.32409	2.14E-05
15	3	4	59.76376	1.33E-05
15	3	5	94.74872	1.57E-05
20	2	2	8.510329	9.6E-06
20	2	3	21.996086	1.1E-05
20	2	4	39.56823	2.36E-05
20	2	5	61.239292	1.54E-05
20	3	2	11.80031	9.2E-06
20	3	3	30.87266	1.04E-05
20	3	4	56.53473	1.45E-05
20	3	5	92.82975	1.67E-05
30	2	2	11.657144	9.1E-06
30	2	3	28.9443	1.38E-05
30	2	4	52.77117	1.39E-05
30	2	5	80.91024	2.33E-05
30	3	2	13.073737	1.29E-05
30	3	3	33.842262	1.15E-05
30	3	4	63.51842	3.38E-05
30	3	5	99.50885	4.43E-05

3 Búsqueda Local

3.1 Pseudocódigo

```
function SWAP(vectors)
    solution ← empty
    best ← vectors
    for vector1 in vectors do
        for vector2 in vectors do
            if vector1 == vector2 then continue
            end if
            solution ← swap(vectors, vector1, vector2)
            if dist(solution) > dist(vectors) then
                best ← solution
            end if
        end for
    end for
    return best
end function
```

3.2 Explicación

Para hacer una búsqueda con este problema de máxima dispersión, se tiene que probar con cada vector de dentro de la solución y fuera e intercambiarlos, para siempre tener el mismo tamaño en la solución. Si la nueva solución mejora la anterior, se actualiza.

4 GRASP

4.1 Pseudocódigo

function GRASP(vectors, lenSol)
  $bestSolution \leftarrow empty$
  for $i = 0,\ i \leq maxIter,\ i++$ **do**
    $currentSolution \leftarrow randomGreedy(vectors, lenSol)$
    $currentSolution \leftarrow localSearch(currentSolution)$
    if $distance(currentSolution) > distance(bestSolution)$ **then**
      $bestSolution \leftarrow currentSolution$
    end if
  end for
  return solution
end function

4.2 Explicación

Este algoritmo es aproximado y se basa en ir creando solutiones aleatorias con un grado de voraz ya que la construcción de soluciones se ejecuta con un Greedy pero con una lista de candidatos. Luego ejecuta una búsqueda local por cada solución construida y si ese mínimo local es mejor que la mejor solución obtenida, se actualiza la mejor solución. En el peor caso este algoritmo va a obtener una solución Greedy, y en el mejor caso el óptimo global, pero en el caso promedio obtiene óptimos locales. El tiempo de ejecución de este algoritmo depende del número de iteraciones.

4.3 Resultados

N Vect	Dim Vect	LenSol	Iter	Cand	Disp	Time
15	2	2	10	2	11.859216	0.0024494
15	2	2	10	3	11.859216	0.0003244
15	2	2	20	2	11.859216	0.0008019
15	2	2	20	3	11.859216	0.000866
15	2	3	10	2	27.372702	0.0009598
15	2	3	10	3	27.3727	0.0004899
15	2	3	20	2	27.372702	0.0008804
15	2	3	20	3	27.372702	0.001544
15	2	4	10	2	49.33072	0.0005568
15	2	4	10	3	49.546158	0.0006292
15	2	4	20	2	49.546158	0.0009798
15	2	4	20	3	49.826782	0.0009873
15	2	5	10	2	78.18862	0.0006644
15	2	5	10	3	79.12954	0.000595
15	2	5	20	2	78.188614	0.0012718

15	2	5	20	3	79.12953	0.0012221
15	3	2	10	2	13.27324	0.0004201
15	3	2	10	3	13.273241	0.0005339
15	3	2	20	2	13.27324	0.0009421
15	3	2	20	3	13.273241	0.0011683
15	3	3	10	2	31.86853	0.0005033
15	3	3	10	3	31.86853	0.0005105
15	3	3	20	2	31.86853	0.0013879
15	3	3	20	3	31.86853	0.0012263
15	3	4	10	2	59.76376	0.0005949
15	3	4	10	3	59.763767	0.0005564
15	3	4	20	2	59.763767	0.001018
15	3	4	20	3	59.763767	0.0010501
15	3	5	10	2	96.08583	0.0006632
15	3	5	10	3	96.08584	0.0007676
15	3	5	20	2	96.08583	0.0014992
15	3	5	20	3	96.08584	0.0020841
20	2	2	10	2	8.510329	0.0008865
20	2	2	10	3	8.51033	0.0004801
20	2	2	20	2	8.510329	0.0010764
20	2	2	20	3	8.51033	0.0013994
20	2	3	10	2	21.99609	0.0007241
20	2	3	10	3	21.99609	0.0007954
20	2	3	20	2	21.99609	0.0013663
20	2	3	20	3	21.99609	0.0012872
20	2	4	10	2	40.00226	0.0010675
20	2	4	10	3	40.002266	0.0007755
20	2	4	20	2	40.00226	0.001865
20	2	4	20	3	40.00226	0.0028284
20	2	5	10	2	63.65169	0.0009598
20	2	5	10	3	62.87286	0.0012346
20	2	5	20	2	63.651676	0.0022852
20	2	5	20	3	63.65169	0.0024158
20	3	2	10	2	11.80031	0.0004463
20	3	2	10	3	11.800311	0.000459
20	3	2	20	2	11.80031	0.0013346
20	3	2	20	3	11.800311	0.0008997
20	3	3	10	2	30.872665	0.0009338
20	3	3	10	3	30.872662	0.0012967
20	3	3	20	2	30.872665	0.0016366
20	3	3	20	3	30.872663	0.0014863
20	3	4	10	2	56.690315	0.00111
20	3	4	10	3	56.690315	0.0007366
20	3	4	20	2	56.690315	0.0020295
20	3	4	20	3	56.690315	0.0014723
20	3	5	10	2	92.82975	0.0007791

20	3	5	10	3	92.82975	0.0010019
20	3	5	20	2	92.82975	0.0021867
20	3	5	20	3	92.82975	0.001931
30	2	2	10	2	11.657144	0.0012108
30	2	2	10	3	11.657144	0.0008509
30	2	2	20	2	11.657144	0.00162
30	2	2	20	3	11.657144	0.0020397
30	2	3	10	2	28.9443	0.0016385
30	2	3	10	3	28.9443	0.0010142
30	2	3	20	2	28.9443	0.0021831
30	2	3	20	3	28.9443	0.0020296
30	2	4	10	2	52.77117	0.0016678
30	2	4	10	3	52.77117	0.002344
30	2	4	20	2	52.77117	0.0025371
30	2	4	20	3	52.77117	0.0039321
30	2	5	10	2	80.91024	0.0024233
30	2	5	10	3	80.91025	0.0016771
30	2	5	20	2	80.91025	0.0045749
30	2	5	20	3	80.91025	0.0040172
30	3	2	10	2	13.073737	0.001373
30	3	2	10	3	13.073737	0.0008502
30	3	2	20	2	13.073737	0.0019316
30	3	2	20	3	13.073738	0.0017813
30	3	3	10	2	34.29053	0.0017334
30	3	3	10	3	34.29053	0.0012216
30	3	3	20	2	34.29053	0.0021106
30	3	3	20	3	34.29053	0.0022316
30	3	4	10	2	63.70196	0.001716
30	3	4	10	3	63.70196	0.0018698
30	3	4	20	2	63.701965	0.003027
30	3	4	20	3	63.70196	0.0034923
30	3	5	10	2	99.59206	0.0015584
30	3	5	10	3	99.59204	0.0021872
30	3	5	20	2	99.59206	0.0032726
30	3	5	20	3	99.59206	0.0033174

4.4 Conclusiones

Es un algoritmo bastante bueno, en algunos casos llega a óptimos locales, en otros no mejora respecto al greedy pero lo más importante es que es bastante rápido, con pocas iteraciones alcanza buenas soluciones y en poco más tiempo que el greedy.

5 Tabu

5.1 Pseudocódigo

```
function TABU(vectors, lenSol, maxList)
    list ← empty
    solution ← Greedy(vectors, lenSol)
    list ← solution
    bestSol ← solution
    for i = 0, i ≤ maxIter, i++ do
        neighbors ← localSearch(bestSol)
        bestN ← neighbors[0]
        for neighbor in neighbors do
            if list.Contains(neighbor) then continue
            end if
            if dist(neighbor) > dist(bestN) then
                bestN ← neighbor
            end if
        end for
        if distance(bestN) > distance(solution) then
            solution ← bestN
        end if
        list ← bestN
        if len(list) > maxList then
            list.RemoveFirst()
        end if
    end for
    return solution
end function
```

5.2 Explicación

El algoritmo Tabu es aproximado como GRASP, pero no es aleatorio. Crea una solución inicial Greedy y a través de esta genera todo su entorno de soluciones y guarda en la lista las que no estén ya en la lista, si alguna es mejor que la solución actual, la actualiza. En este caso se está tratando con un algoritmo Tabu con memoria a corto plazo por lo que una vez que la lista llega a su máximo de capacidad se va eliminando la primera solución de la lista.

5.3 Resultados

N Vect	Dim Vect	LenSol	Iter	Cand	Disp	Time
15	2	2	10	2	11.859216	0.0011094
15	2	2	10	3	11.859216	0.000257
15	2	2	20	2	11.859216	0.0004776
15	2	2	20	3	11.859216	0.0004286
15	2	3	10	2	27.372704	0.0005227
15	2	3	10	3	27.372704	0.0004557
15	2	3	20	2	27.372704	0.0006527
15	2	3	20	3	27.372704	0.0005318
15	2	4	10	2	49.826782	0.0003822
15	2	4	10	3	49.826782	0.0004482
15	2	4	20	2	49.826782	0.0007125
15	2	4	20	3	49.826782	0.0008925
15	2	5	10	2	79.12953	0.0004028
15	2	5	10	3	79.12953	0.0006188
15	2	5	20	2	79.12953	0.0014192
15	2	5	20	3	79.12953	0.0008835
15	3	2	10	2	13.27324	0.0004105
15	3	2	10	3	13.27324	0.0005105
15	3	2	20	2	13.27324	0.0005325
15	3	2	20	3	13.27324	0.0004124
15	3	3	10	2	31.868534	0.0003152
15	3	3	10	3	31.868534	0.0003535
15	3	3	20	2	31.868534	0.0008376
15	3	3	20	3	31.868534	0.0010192
15	3	4	10	2	59.76376	0.0003831
15	3	4	10	3	59.76376	0.0005367
15	3	4	20	2	59.76376	0.0011435
15	3	4	20	3	59.76376	0.0006162
15	3	5	10	2	96.08583	0.0003811
15	3	5	10	3	96.08583	0.0007464
15	3	5	20	2	96.08583	0.0010818
15	3	5	20	3	96.08583	0.0008189
20	2	2	10	2	8.510329	0.0005959
20	2	2	10	3	8.510329	0.0004614
20	2	2	20	2	8.510329	0.0006873
20	2	2	20	3	8.510329	0.0006814
20	2	3	10	2	21.996086	0.0005401
20	2	3	10	3	21.996086	0.0005411
20	2	3	20	2	21.996086	0.0009099
20	2	3	20	3	21.996086	0.0012876
20	2	4	10	2	40.00226	0.0006496
20	2	4	10	3	40.00226	0.0007171

20	2	4	20	2	40.00226	0.0015232
20	2	4	20	3	40.00226	0.0013634
20	2	5	10	2	63.651676	0.0010727
20	2	5	10	3	63.651676	0.0012915
20	2	5	20	2	63.651676	0.0020025
20	2	5	20	3	63.651676	0.0018135
20	3	2	10	2	11.80031	0.00033
20	3	2	10	3	11.80031	0.0006266
20	3	2	20	2	11.80031	0.0011265
20	3	2	20	3	11.80031	0.00089
20	3	3	10	2	30.872662	0.00083
20	3	3	10	3	30.872662	0.0010918
20	3	3	20	2	30.872662	0.0010691
20	3	3	20	3	30.872662	0.0008797
20	3	4	10	2	56.690323	0.0011069
20	3	4	10	3	56.690323	0.0010321
20	3	4	20	2	56.690323	0.0013569
20	3	4	20	3	56.690323	0.0011164
20	3	5	10	2	92.82975	0.0007875
20	3	5	10	3	92.82975	0.0007521
20	3	5	20	2	92.82975	0.0012627
20	3	5	20	3	92.82975	0.0017108
30	2	2	10	2	11.657144	0.0014635
30	2	2	10	3	11.657144	0.0008183
30	2	2	20	2	11.657144	0.0013246
30	2	2	20	3	11.657144	0.0016163
30	2	3	10	2	28.9443	0.0009019
30	2	3	10	3	28.9443	0.0015479
30	2	3	20	2	28.9443	0.0030693
30	2	3	20	3	28.9443	0.0020566
30	2	4	10	2	52.77117	0.0012167
30	2	4	10	3	52.77117	0.0013627
30	2	4	20	2	52.77117	0.0036732
30	2	4	20	3	52.77117	0.0026603
30	2	5	10	2	80.91024	0.0026003
30	2	5	10	3	80.91024	0.0015759
30	2	5	20	2	80.91024	0.0035599
30	2	5	20	3	80.91024	0.0032714
30	3	2	10	2	13.073737	0.0006648
30	3	2	10	3	13.073737	0.0008136
30	3	2	20	2	13.073737	0.0013727
30	3	2	20	3	13.073737	0.0017802
30	3	3	10	2	34.290527	0.0009728
30	3	3	10	3	34.290527	0.0009148
30	3	3	20	2	34.290527	0.0023751
30	3	3	20	3	34.290527	0.0021231

30	3	4	10	2	63.701965	0.0011648
30	3	4	10	3	63.701965	0.0011825
30	3	4	20	2	63.701965	0.0023444
30	3	4	20	3	63.701965	0.00245
30	3	5	10	2	99.59206	0.0014178
30	3	5	10	3	99.59206	0.001607
30	3	5	20	2	99.59206	0.0028817
30	3	5	20	3	99.59206	0.0029295

5.4 Conclusiones

Es un algoritmo rinde bastante igual que el grasp, inclusive en los resultados en la mayoría de casos obtiene mejores resultados y mejores tiempos, pero no siempre, en otros casos grasp obtiene mejores resultados, depende del tipo de problema a solucionar.

6 Ramificación y Poda

6.1 Pseudocódigo

function BrunchBound(vectors, lenSol, initialGenerator, typeStrategy)
    $solution \leftarrow initialGenerator(vectors, lenSol)$
    $lowerBound \leftarrow dist(solution)$
    $list \leftarrow Candidates(solution)$
    while *list not empty* **do**
        if $typeStrategy == depth\ first$ **then**
            $current \leftarrow list.First()$
        else $typeStrategy == lowest\ bound$
            $current \leftarrow lowest(list)$
        end if
        $list \leftarrow Candidates(current)$
    end while
    return solution
end function

6.2 Explicación

El algoritmo de Ramificación y Poda es exacto como el Greedy, pero al contrario que este, el de ramificación y poda si genera soluciones óptimas. Se basa en utilizar una solución inicial, ya sea con Greedy o Grasp por ejemplo y a partir de ahí marcar que como mínimo la mejor solución será igual que la solución inicial, creamos las ramas iniciales y las metemos en una lista. La creación de ramas tiene que ser algo optimista porque vamos a utilizar una cota superior, por lo que si una rama puede llegar como máximo a una cota superior inferior a nuestra cota inicial, podemos descartar esa rama. Luego, mientras la lista no esté vacia, escogemos una rama de la lista y generamos sus ramas, comprobando que sus cotas no sean inferior y así hasta llegar a soluciones, una vez tengamos soluciones verificamos si su coste es mejor que el inicial y actualizamos la cota inferior.

6.3 Resultados, Greedy - Nodo con cota superior menor

N Vect	Dim Vect	LenSol	Disp	Time	Nodes
15	2	2	11.859216	0.0028669	0
15	2	3	27.3727	0.0024542	161
15	2	4	49.826782	0.0205516	2343
15	2	5	79.12953	0.2626457	27190
15	3	2	13.27324	0.0001377	0
15	3	3	31.868526	0.0035078	154
15	3	4	59.76376	0.0233726	1575
15	3	5	96.08584	0.1623628	13677
20	2	2	8.510329	0.0001659	0
20	2	3	21.996086	0.0035149	156
20	2	4	40.00226	0.0394617	3055
20	2	5	63.65168	0.7303447	39965
20	3	2	11.80031	0.0001933	0
20	3	3	30.87266	0.0046004	136
20	3	4	56.690315	0.0378945	2010
20	3	5	92.82976	0.4393323	13828
30	2	2	11.657144	0.0006839	0
30	2	3	28.9443	0.0253883	302
30	2	4	52.77117	0.4749125	5148
30	2	5	80.91025	8.5867668	141852
30	3	2	13.073737	0.0005544	0
30	3	3	34.290527	0.0115539	248
30	3	4	63.70196	0.1502686	3940
30	3	5	99.59205	2.9630423	88346

6.4 Resultados, Grasp - Nodo con cota superior menor

N Vect	Dim Vect	LenSol	Disp	Time	Nodes
15	2	2	11.859216	0.0016888	0
15	2	3	27.372702	0.0033079	159
15	2	4	49.826782	0.0244418	2328
15	2	5	79.12953	0.2793821	27123
15	3	2	13.273241	0.0009177	0
15	3	3	31.86853	0.0022354	141
15	3	4	59.763763	0.01543	1575
15	3	5	96.08584	0.104839	13101
20	2	2	8.51033	0.0075727	0
20	2	3	21.996086	0.0065291	156
20	2	4	40.002266	0.0613034	3040
20	2	5	63.651688	0.930975	38890
20	3	2	11.80031	0.0024208	0
20	3	3	30.872663	0.0074982	136
20	3	4	56.690315	0.0680641	2002
20	3	5	92.82976	0.468785	13828
30	2	2	11.657144	0.0030477	0
30	2	3	28.9443	0.0136549	302
30	2	4	52.771175	0.179276	5148
30	2	5	80.91025	5.5641726	141852
30	3	2	13.073738	0.009915	0
30	3	3	34.290527	0.025441	236
30	3	4	63.70196	0.2023478	3896
30	3	5	99.59206	2.6298684	88080

6.5 Resultados, Greedy - Búsqueda en Profundidad

N Vect	Dim Vect	LenSol	Disp	Time	Nodes
15	2	2	11.859216	0.0492837	0
15	2	3	27.3727	0.0052738	159
15	2	4	49.826782	0.038139	2329
15	2	5	79.12953	0.3692674	27124
15	3	2	13.27324	8.53e-05	0
15	3	3	31.868526	0.0013271	154
15	3	4	59.76376	0.014319	1575
15	3	5	96.08584	0.2116309	13733
20	2	2	8.510329	0.0003919	0
20	2	3	21.996086	0.0089225	156
20	2	4	40.00226	0.1194285	3049
20	2	5	63.65168	1.1596974	39900
20	3	2	11.80031	0.0001734	0
20	3	3	30.87266	0.0061889	136
20	3	4	56.690315	0.0523057	2011
20	3	5	92.82976	0.453384	13828
30	2	2	11.657144	0.0005764	0
30	2	3	28.9443	0.0107232	302
30	2	4	52.77117	0.1795688	5148
30	2	5	80.91025	4.9800197	141852
30	3	2	13.073737	0.0003575	0
30	3	3	34.290527	0.0094282	248
30	3	4	63.70196	0.1816335	3941
30	3	5	99.59205	3.5886891	88394

6.6 Resultados, Grasp - Búsqueda en Profundidad

N Vect	Dim Vect	LenSol	Disp	Time	Nodes
15	2	2	11.859216	0.0008908	0
15	2	3	27.372702	0.00404	159
15	2	4	49.826782	0.0333566	2325
15	2	5	79.12953	0.3593822	27123
15	3	2	13.27324	0.0023449	0
15	3	3	31.86853	0.0040394	141
15	3	4	59.763763	0.0245681	1575
15	3	5	96.08584	0.1948304	13101
20	2	2	8.510329	0.001626	0
20	2	3	21.996088	0.0043173	156
20	2	4	40.00226	0.0681161	3040
20	2	5	63.65168	1.1775794	38890
20	3	2	11.80031	0.0028384	0
20	3	3	30.872663	0.0102273	136
20	3	4	56.690315	0.0971992	2002
20	3	5	92.82976	0.6847822	13828
30	2	2	11.657144	0.0040612	0
30	2	3	28.9443	0.0151938	302
30	2	4	52.771183	0.1789156	5148
30	2	5	80.91025	4.9671429	141852
30	3	2	13.073738	0.0037956	0
30	3	3	34.290527	0.0145825	236
30	3	4	63.701965	0.2860887	3896
30	3	5	99.59206	3.7590269	88080

6.7 Conclusiones

Utilizar Grasp como método inicializador mejora un poco en los resultados, aunque no mucho y por lo general la estrategia de búsqueda en profundidad suele ser más rápido, pero en determinados casos no.