|  |
| --- |
| **Rapport du Mini-Projet CSP**  **Master 2 IA, 2020/2021** |

Nom et Prénom du Binôme :

Duc Viet NGUYEN

Benoit ....

Ex1:

* Description: Le problème de la distribution de fréquence: Les Transmitters sont divisés en cellules, chaque cellule possède N Transmitters. La fréquence distribuée satisfait les conditions suivantes :
  + Dans la même cellule, chaque fréquence de Transmitters doit être espacée d'au moins 16
  + Chaque cellule est séparée d'une distance D, la fréquence de Transmitter des deux cellules doit être espacée d'au moins D
* Modele:
  + Variable:
    - freqs: une matrice avec dim(freqs) = (nbrCell, maxNbrTrans) à présenter distribution de fréquence. (value = 0 ⬄ il y a pas transporter)

exemple:

[1 17 33 49 65 81 97 113 0 0

2 18 34 50 66 82 0 0 0 0

4 20 36 52 68 84 0 0 0 0

1 0 0 0 0 0 0 0 0 0

3 19 35 51 0 0 0 0 0 0

5 21 37 53 0 0 0 0 0 0

1 17 33 49 65 81 97 113 0 0

3 19 35 51 67 83 99 115 0 0

6 22 38 54 70 86 102 118 0 0

8 24 40 56 72 88 104 120 0 0

10 26 42 58 0 0 0 0 0 0

12 28 44 60 76 92 108 124 140 0

14 30 46 62 78 94 110 126 0 0

74 90 106 122 0 0 0 0 0 0

99 115 131 147 0 0 0 0 0 0

5 21 37 53 69 85 101 117 133 149

7 23 39 55 71 87 103 119 0 0

2 18 34 50 66 82 98 114 130 0

9 25 41 57 73 89 105 121 0 0

128 144 160 176 0 0 0 0 0 0

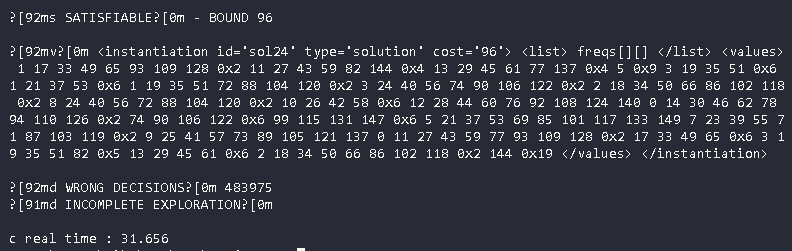
3 19 35 51 67 0 0 0 0 0

11 27 43 59 0 0 0 0 0 0

2 18 34 50 66 82 98 114 0 0

13 0 0 0 0 0 0 0 0 0

15 0 0 0 0 0 0 0 0 0]

* + contraintes:
    - Il n’y a pas de transporter ⬄ value = 0
    - Meme cell, abs(freqs[i] – freqs[j]) >= 16
    - Entrer 2 cell h et j: abs(freqs[k][i] – freqs[h][j]) >= distance(h,j)
  + Minimiser:
    - NValues(freqs): minimiser le nombre de freqs qui sont utilisés.
* Probleme:
  + La solving dure longtemps, mais n'a pas encore trouvé le résultat optimal -> solution: exécuter la solving pendant un temps limité, obtenir le résultat acceptable.
* Résultat:
  + Chaque ligne est resulta de chaque cell:

1 17 33 49 65 93 109 128 0x2

11 27 43 59 82 144 0x4

13 29 45 61 77 137 0x4

5 0x9

3 19 35 51 0x6

1 21 37 53 0x6

1 19 35 51 72 88 104 120 0x2

3 24 40 56 74 90 106 122 0x2

2 18 34 50 66 86 102 118 0x2

8 24 40 56 72 88 104 120 0x2

10 26 42 58 0x6

12 28 44 60 76 92 108 124 140 0

14 30 46 62 78 94 110 126 0x2

74 90 106 122 0x6

99 115 131 147 0x6

5 21 37 53 69 85 101 117 133 149

7 23 39 55 71 87 103 119 0x2

9 25 41 57 73 89 105 121 137 0

11 27 43 59 77 93 109 128 0x2

17 33 49 65 0x6

3 19 35 51 82 0x5

13 29 45 61 0x6

2 18 34 50 66 86 102 118 0x2

144 0x19

* + **Nombre de frequence utilisé est 95** (BOUND 96 dans la résultat est compté 0)

Ex2:

* Description: Distribuer les tests aux machines appropriées afin que le temps pour effectuer tous les tests soit le plus court. Certains tests ne sont disponibles que sur certaines machines. Certains tests nécessitent des ressources, tandis que d'autres tests utilisant la même ressource ne peuvent pas être exécutés en même temps.
* Modele:
  + Variables:
    - m[i] est nom de machine appliquer pour test i-eme

exemple: m: [0, 0, 1, 1, 0, 2, 0, 1, 2, 2]

* + - p[i] = [debut,fin] est le début et la fin de chaque test

exemple: p: [

[4, 6],

[7, 11],

[0, 3],

[3, 7],

[0, 3],

[0, 2],

[3, 4],

[7, 9],

[7, 10],

[2, 7]

]

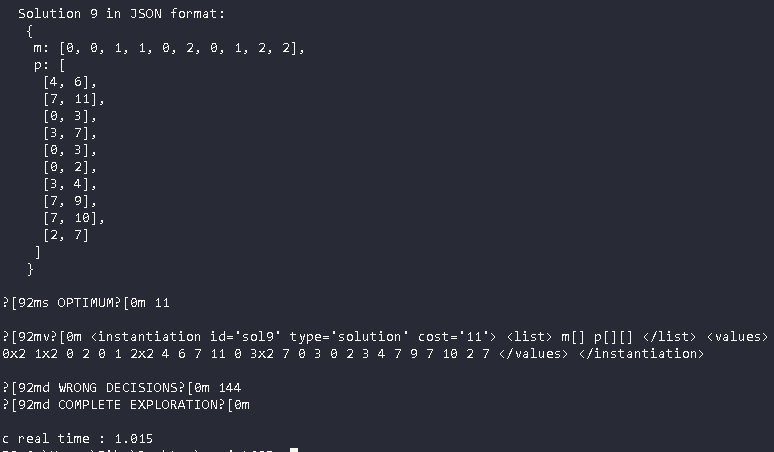
* contraintes:
  + Certains tests ne s'exécutent que sur certaines machines: les noms de ces machines sont stockés dans les tests[i][1] pour test i-eme
  + La fin (fin) = le debut (debut) + temps d'exécution du test (test[i][0])
  + Pour chaque machine, un seul test peut être effectué à la fois: debut[i]>=fin[j] or debut[j]>=fin[i]
  + Pour chaque ressource, il n'est possible de fournir qu'un seul test à la fois:

p[resource[h][i]][0] >= p[resource[h][j]][1]

or p[resource[h][i]][1] <= p[resource[h][j]][0]

* Minimiser:
  + Max(finTemp de tous les tests): Minimiser (le maximum de la fin de tous les tests).
* Probleme:
  + Changer la formule de la matrice qui présente une distribution de ressource:

Resource[h] = [test[i],test[j], etc…] : resource h est utilisé pour test I, test j, etc…

* + Pour les petites données, solving marche bien, le temps d’exec est court. Pour les grandes données, le temps d’exec est trop long, alors on doit limiter le temps de solving et prendre le meilleur résultats.
* Resultat:
  + java -jar ACE-21-01.jar testMachine-t10-example.xml

Solution 9 in JSON format:

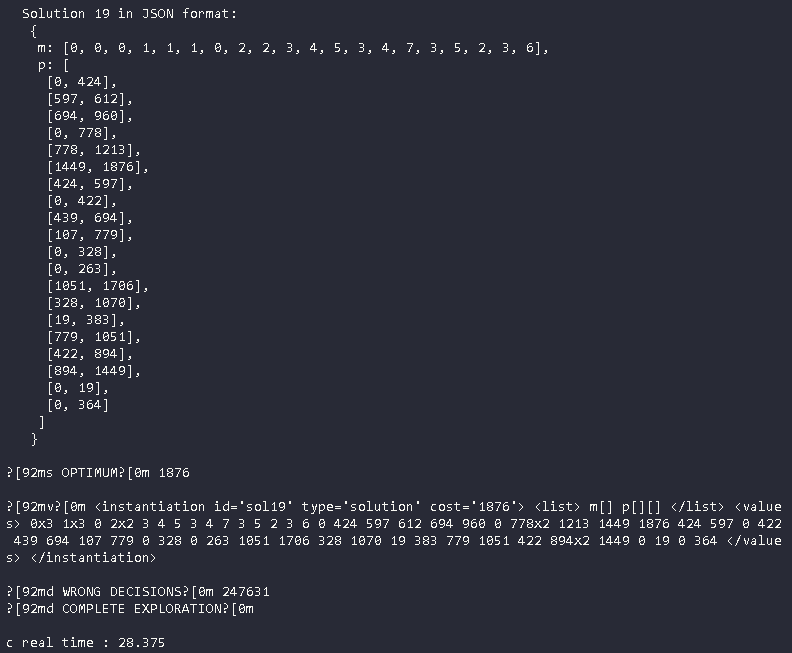
{

m: [0, 0, 1, 1, 0, 2, 0, 1, 2, 2],

p: [[4, 6], [7, 11], [0, 3], [3, 7], [0, 3], [0, 2], [3, 4], [7, 9], [7, 10], [2, 7] ]

}

**Temps pour faire tous les tests est 11**

* + java -jar ACE-21-01.jar testMachine-t20m10r3-1.xml

Solution 19 in JSON format:

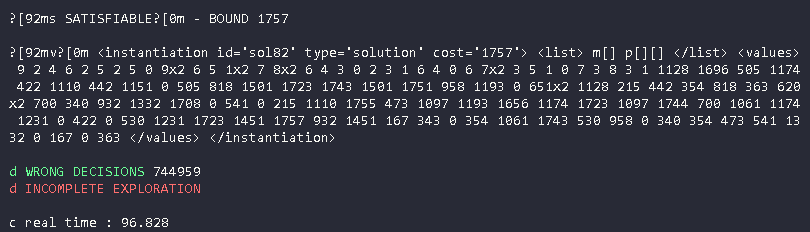
{

m: [0, 0, 0, 1, 1, 1, 0, 2, 2, 3, 4, 5, 3, 4, 7, 3, 5, 2, 3, 6],

p: [ [0, 424], [597, 612], [694, 960], [0, 778], [778, 1213], [1449, 1876], [424, 597], [0, 422], [439, 694], [107, 779], [0, 328], [0, 263], [1051, 1706], [328, 1070], [19, 383], [779, 1051], [422, 894], [894, 1449], [0, 19], [0, 364]]

}

**Temps pour faire tous les tests est 1876**

* + java -jar ACE-21-01.jar testMachine-t40m10r3-2.xml

m: [9, 2, 4, 6, 2, 5, 2, 5, 0, 9, 9, 6, 5, 1, 1, 7, 8, 8, 6, 4, 3, 0, 2, 3, 1, 6, 4, 0, 6, 7, 7, 3, 5, 1, 0, 7, 3, 8, 3, 1]

p: [[1128, 1696], [505, 1174], [422, 1110], [442, 1151], [0, 505], [818, 1501], [1723, 1743], [1501, 1751], [958, 1193], [0, 651], [651, 1128], [215, 442], [354, 818], [363, 620], [620, 700], [340, 932], [1332, 1708], [0, 541], [0, 215], [1110, 1755], [473, 1097], [1193, 1656], [1174, 1723], [1097, 1744], [700, 1061], [1174, 1231], [0, 422], [0, 530], [1231, 1723], [1451, 1757], [932, 1451], [167, 343], [0, 354], [1061, 1743], [530, 958], [0, 340], [354, 473], [541, 1332], [0, 167], [0, 363]]

**Temps pour faire tous les tests est 1757**