Rapport de projet OS10

Etude de l'algorithme GRASP pour le problème d'ordonnancement FJSSP-nfa

BOUCETTA I. & TRAN Q.N.H. & WARTELLE A.

7 janvier 2019

Sommaire

Ce travail a pour objectif d'étudier l'article de 2010 de Rajkumar, Muthukannan & Asokan, P & Vamsikrishna, V. portant sur un algorithme de recherche locale randomisée : *Greedy Randomized Adaptive Search Procedure* (GRASP), dans le cadre d'un problème de job shop : le *Flexible Job Shop Scheduling Problem with non fixed-availability constraints* (FJSSP-nfa) [1]. Cette extension du problème du job shop permet de prendre en compte deux aspects importants de la production :

1. la flexibilité des machines pour effectuer des opérations, c'est-à-dire qu'une opération n'aura pas une machine prédéfinie à l'avance pour son exécution. On aura en effet le choix entre plusieurs machines avec différents temps d'exécution associés. Cette diversité des temps d'exécution est en général lié à la performance et à l'adéquation de chaque machine pour exécuter une tâche
2. la prise en compte de tâches de maintenance à effectuer dans des fenêtres de temps données (indisponibilité non fixe). En effet les machines ont besoin d'être entretenues afin d'éviter (si possible) des pannes ou des défaillances. Durant une tâche de maintenance, une machine et indisponible est ne peut pas traiter d'opération, ce qui impacte fortement les performances d'un choix d'affectation d'opérations et d'ordonnancement de celles-ci.

Avec la prise en compte de la flexibilité des machines, le FJSSP-nfa permet de travailler sur de nouveaux critères portant sur la charge de travail, c'est à dire sur le temps d'utilisation de chaque machine qui n'est plus fixé comme dans le cas d'un job shop classique (CJSSP). Ainsi le FJSSP-nfa est un problème multi-critère portant sur la charge de travail totale et maximal des machines et sur le makespan de l'ordonnancement (temps de complétion maximale).

Le problème de job shop classique étant déjà NP-difficile, le FJSSP-nfa l'est aussi. De plus, avec les difficultés rajoutées par la flexibilité des machines et la maintenance, une approche de résolution par méthode exacte est presque inenvisageable [1]. Ainsi ce sont des heuristiques comme l'algorithme de recherche locale GRASP ([1]) et des méta-heuristiques comme les algorithmes génétiques([2],[3]) qui rencontrent le plus de succès pour ce type de problème car elle permettent d'avoir de bons résultats multi-critères en un temps raisonnable.

1 Définitions : FJSSP-nfa et GRASP

Avec la notation de Graham, le problème d'ordonnancement FJSSP-nfa peut s'écrire sous la forme :

$$J_m|flexibility, nfa|W_{tot}, W_{max}, C_{max}$$

1. d'après [2], les méthodes exactes par graphes disjonctifs sont actuellement limités à 20 jobs et 10 machines pour le FJSSP (sans maintenance)

Plus précisément, on souhaite minimiser une combinaison linéaire de W_{tot} (charge totale d'occupation des machines),W_{max} (charge maximale d'occupation des machines) et C_{max} : $W_1 * W_{tot} + W_2 * W_{max} + W_3 * C_{max}$ avec $W_1 + W_2 + W_3 = 1$ et W_1,W_2,W_3 des paramètres donnés en fonction des préférences d'objectif.

En reprenant les notations de l'article [1], on a n jobs d'incice i avec n_i opérations d'indice k pour chaque job i (notées O_{ik}), m machine d'indice j, L_j tâches de maintenance d'indice l pour chaque machine j (notées PM_{jl}). Chaque opération peut être affectées sur un ensemble A_{ik} de machines avec différentes durées t_{ikj} d'exécution. Les tâches de maintenance ont des durées p_{jl} d'exécution et doivent se terminées sur des fenêtres de temps $[t^E_{jl}, t^L_{jl}]$ (E pour "Early" et L pour "Late").

L'algorithme GRASP est un algorithme de recherche locale randomisée avec une approche hiérarchique pour résoudre le FJSSP-nfa, c'est-à-dire qu'il va d'abord résoudre le problème d'affectation des opérations O_{ik} aux machines de A_{ik} ($\forall i, k$) puis le problème d'ordonnancement des opérations O_{ik} affectées et des tâches de maintenance PM_{jl} sur chaque machine [2]. Comme le montre les algorithmes 1 et 2 (voir aussi [1]), GRASP va construire une solution en ajoutant itérativement les opérations [3] (pour l'affectation et l'ordonnancement) qui sont choisies de manière aléatoire dans une liste RCL de candidats restreinte.

La RCL est un voisinage de l'opération qui fait augmenter le moins la valeur de la fonction objective (W_{tot} pour l'algorithme 1 et C_{max} pour l'algorithme 2). La RCL est triée par valeur croissante des augmentations de la valeur objective afin de pondérer la probabilité de choisir chaque candidat (opération à affecter) en fonction de son rang r avec la fonction $r \to 1/r$.

La taille de la RCL est paramétrée par une valeur $\alpha \in [0, 1]$ qui va permettre de fixer la valeur maximale d'augmentation de la fonction objective (pour être acceptée dans la liste) par $OFV_{min} + \alpha * (OFV_{max} - OFV_{min})$ avec OFV_{min} et OFV_{max} les valeurs minimales et maximales (respectivement) de la fonction objective après l'ajout de la prochaine opération.

Finalement, pour chaque itération principale de l'algorithme, on sélectionne aléatoirement (avec un générateur (pseudo-)aléatoire de loi uniforme) une valeur $\alpha \in [0, 1]$ qui sera utilisée pour construire une solution complète au problème de FJSSP-nfa. On obtient ainsi une ensemble de solutions où chacune indique les affectations $x_{ikj} \in \{0, 1\}$ (ou $x_{ik} \in \{0, 1..., j\}$ dans notre algorithme) et les temps de complétions c_{ik} et y_{jl} des opérations O_{ik} et des tâches de maintenances PM_{jl} respectivement. On sélectionne enfin la solution qui minimise la fonction objective $W_1 * W_{tot} + W_2 * W_{max} + W_3 * C_{max}$.

2. L'approche hiérarchique s'oppose à l'approche intrinsèque où les deux problèmes sont résolus conjointement, cette approche est plus difficile mais peut donner de meilleurs résultats notamment avec les algorithmes génétiques

3. les tâches de maintenance sont ajoutées par des détections de conflit avec l'ordonnancement d'uen opération et lorsque toutes les opérations ont pu être ordonnancées

Algorithme 1 : Assigner les opérations aux machines (*routing*)

`// Initialisation`

t le nombre de solutions;

$A = \{\alpha_1, \alpha_2, ..., \alpha_t\}$ l'ensemble des valeurs α discrètes aléatoires telle que $0 \leq \alpha_i \leq 1 \, \forall i = \overline{1, t}$;

$S = \{s_1, s_2, ..., s_t\}$ l'ensemble des solutions à construire;

`// Assignment`

pour chaque $\alpha_i \in A$ **faire**

  σ l'ensemble des opérations non assignées;

  s_i la solution à construire;

  $s_i \leftarrow \emptyset$;

  tant que $\sigma \neq \emptyset$ **faire**

    `// Le processing time` t_o `dépend de la machine de traitement et correspond à l'augmentation de` W_{tot} `(de` s_i`) par l'affectation d'une opération`

    Chercher O_{Max} l'opération dont le processing time est le plus long;

    Chercher O_{min} l'opréation dont le processing time est le plus court;

    $Range \leftarrow t_{O_{Max}} - t_{O_{min}}$;

    $Width \leftarrow Range * \alpha_i$;

    Construire RCL l'ensemble des opérations non assignées $o \in \sigma$ telle que $t_{O_{min}} \leq t_o \leq t_{O_{min}} + Width$;

    Ordonner RCL dans l'ordre croissant de processing time;

    Pondérer les éléments de RCL par la fonction $\frac{1}{r}$ où r est l'ordre de l'élément;

    Choisir de manière aléatoire un élément o^* de RCL en tenant compte des poids ci-dessus (la probabilité de choisir une opération o est $\frac{poids_o}{poids_{total}}$);

    $s_i \leftarrow s_i \cup o^*$;

    $\sigma \leftarrow \sigma \backslash o^*$;

  fin

  Calculer la charge totale de s_i;

fin

Algorithme 2 : Ordonner les opérations et les maintenances sur les machines (*scheduling*)

`// Initialisation`

t le nombre de solutions (fixé à l'algorithme (1)) ;

$A = \{\alpha_1, \alpha_2, ..., \alpha_t\}$ l'ensemble des valeurs α discrètes aléatoires telle que $0 \leq \alpha_i \leq 1 \, \forall i = \overline{1,t}$ (pas forcément identique à l'algorithme (1));

$S = \{s_1, s_2, ..., s_t\}$ l'ensemble des solutions à construire;

`// Ordonnancement`

pour chaque $\alpha_i \in A$ **faire**

  σ l'ensemble des opérations non ordonnées;

  s_i la solution à ordonner;

  $s_i \leftarrow \emptyset$;

  tant que $\sigma \neq \emptyset$ **faire**

    `// Le processing time dépend de machine de traitement`

    Chercher o_{Max} l'opération qui augmente le plus C_{max} (de s_i) ; et o_{min} l'opération qui augmente le moins C_{max} en tenant compte les maintenances dont les temps de complétion doivent se situer dans les intervalles donnés $\left[t^E_{jl}, t^L_{jl}\right]$;

    $Range \leftarrow C^{o_{Max}}_{max} - C^{o_{min}}_{max}$;

    $Width \leftarrow 1 + Range * \alpha_i$;

    Construire RCL l'ensemble des opérations non assignées $o \in \sigma$ telle que $C^{o_{min}}_{max} \leq C^{o}_{max} \leq C^{o_{min}}_{max} + Width$;

    Ordonner RCL dans l'ordre croissant des C_{max};

    Pondérer les éléments de RCL par la fonction $\frac{1}{r}$ où r est l'ordre de l'élément;

    Choisir de manière aléatoire un élément o^* de RCL en tenant compte des poids ci-dessus (la probabilité de choisir une opération o est $\frac{poids_o}{poids_{total}}$;

    Ordonner o^* en tenant compte la date de disponibilité la plus tôt et les maintenances;

    $\sigma \leftarrow \sigma \backslash o^*$;

  fin

  Ordonner les maintenances sans collision à la suite opérations ordonnancées;

  Calculer le makespan (C_{max}) de s_i;

fin

2 Résultats et comparaisons

2.1 Spécification de l'ordinateur

Nous tournons le code sur un ordinateur portable de processeur Intel(R) Core(TM) i7-7700HQ CPU2.8GHz (8 CPUs), 8 Go de RAM. Le programme est compilé par Visual Studio Code 2017. Le générateur choisi est `MersenneTwister.h` (v1.0, 15/05/2003).

2.2 Résultats et comparaisons avec ceux de l'article

Nous réalisons 1000 solutions (itérations) séquentiellement parmi lesquelles nous choissions les meilleurs corresponsdant au triplet voulu.

Critères	Valeurs de l'article	Nos valeurs
W_t	103	103
W_{max}	16	15
C_{max}	18	18
$F(0.5; 0.3; 0.2)$	59.9	59.9
$F(0.5; 0.2; 0.3)$	60.1	60.1

TABLE 1 – Résultat pour problème 8×8

Critères	Valeurs de l'article	Nos valeurs
W_t	60	60
W_{max}	7	8
C_{max}	9	12
$F(0.5; 0.3; 0.2)$	33.9	35.4
$F(0.5; 0.2; 0.3)$	34.1	35.6

TABLE 2 – Résultat pour problème 10×10

Critères	Valeurs de l'article	Nos valeurs
W_t	107	98
W_{max}	13	14
C_{max}	16	19
$F(0.5; 0.3; 0.2)$	60.3	57.2
$F(0.5; 0.2; 0.3)$	60.9	57.8

TABLE 3 – Résultat pour problème 15×10

Nos valeurs sont très proche celles trouvées dans l'articles, particulièrement certaines sont meilleures, signifiant la reproductivité facile de l'algorithme. Son application aura peu de dépendance de l'ordinateur de planification au niveau de résultat.

Critères	Valeurs de l'article	Nos valeurs
W_t	40	40
W_{max}	9	9
C_{max}	16	12
$F(0.5; 0.3; 0.2)$	25.9	25.6
$F(0.5; 0.2; 0.3)$	26.6	25.9

TABLE 4 – Résultat pour problème 4×5

D'autre part, au niveau de temps d'exécution, qui joue un rôle important de chaque algorithme, leurs déroulement ne prennent pas plus de *2 seconds* (figure (5)). Autrement dit, l'algorithme GRASP est efficace contre le problème de type FJSSP-nfa.

Problème	Temps d'exécution (ms)
4×5	28.6767
8×8	150.71
10×10	427.403
15×10	1661.43

TABLE 5 – Temps d'exécution de chaque problème exemplaire

Les ordonnancements finaux pour chaque problème exemplaire sont inclus dans l'annexe (3.3).

2.3 Analyse supplémentaire : frontières de dominance de Pareto

Afin de compléter l'analyse donnée par l'article, nous avons décidé de regarder les frontières de dominance de Pareto sur 5 niveaux (le niveau i dominant le niveau i+1) qui sont disponibles en annexes 3.4. Nous avons observé que, tout en ayant 1000 solutions, les premières frontières sont composées de très peu de vecteurs de solutions (composé des trois valeurs objectives) avec en général seulement 2 pour la première frontière et quelques dizaines au total pour les 5 premières. De plus les solutions sont assez bien distribuées dans l'espace et on a pas trop de vecteurs objectifs (environ 2 ou 3 au maximum) identiques.

On remarque que les objectifs à minimiser sont souvent positivement corrélés, c'est-à-dire que la diminution de l'un implique souvent la diminution d'au moins l'un des autres. Cela semble logique puisque moins de charge de travail permettra souvent aux machines de terminer plus tôt (makespan diminué) et terminer plus tôt nécessite souvent d'avoir moins de charge. Néanmoins l'ajout d'une recherche locale randomisée permet de découvrir des solutions souvent meilleurs et peu évidentes car elles peuvent aller à l'encontre de cette corrélation positive. Ainsi, l'algorithme GRASP semble être une approche adaptée pour résoudre le FJSSP-nfa en multi-objectifs.

3 Annexes

3.1 Visualisation de tournées (plotJob.m)

```
function plotJob(njob,nmachine,type)
close all;

% read input lines
prefix=strcat('results',num2str(njob),'x',num2str(nmachine),'_',type);
f=fopen(strcat('./output/',prefix,'.out'));
tline = fgetl(f);
tlines = cell(0,1);
while ischar(tline)
    tlines{end+1,1} = tline;
    tline = fgetl(f);
end
fclose(f);


n = sscanf(tlines{1}, '%d %d'); %number of jobs and maintenance tasks
OF =  sscanf(tlines{2}, '%f %f %f %f'); %Factors and function objective value
OFVS =  sscanf(tlines{3}, '%f %f %f'); %Objective values
data_jobs = zeros(n(1), 5); %scheduling data of jobs
data_maint = zeros(n(2), 4); %scheduling data of maintenances tasks
for i=1:n(1)
  data_jobs(i,:) = sscanf(tlines{i+3}, '%d %d %d %f %f');
end
for i=1:n(2)
  data_maint(i,:) = sscanf(tlines{i+n(1)+3}, '%d %d %d %f %f');
end

colors = lines(n(1));
figure;
%Plot the bloc of each operation
for i=1:n(1)
  X = data_jobs(i,4);
  Y = data_jobs(i,3);
  W = data_jobs(i,5) - data_jobs(i,4);
  H = 1;
  color = colors(data_jobs(i,1)+1,:) + 0.1*floor((data_jobs(i,1)+1)/8)*ones(1,3);
  rectangle('Position',[X Y W H],'FaceColor',color);
  text(X,Y+H/2, ['O_{',num2str(data_jobs(i,1)+1),',',num2str(data_jobs(i,2)+1),'}'
      ]);
end
%Plot the bloc of each maintenance task
for i=1:n(2)
  X = data_maint(i,3);
  Y = data_maint(i,1);
  W = data_maint(i,4) - data_maint(i,3);
  H = 1;
  color = [0.7 0.7 0.7];
  rectangle('Position',[X Y W H],'FaceColor',color);
  text(X,Y+H/2, ['PM_{',num2str(data_maint(i,1)+1),',',num2str(data_maint(i,2)+1),'
      }']);
end
maintitle = ['W_{tot}=',num2str(OFVS(1)),'  W_{max}=',num2str(OFVS(2)),'  C_{max}='
    ,num2str(OFVS(3))];
subtitle = ['F=',num2str(OF(1)),'*W_{tot}+',num2str(OF(2)),'*W_{max}+',num2str(OF
    (3)),'*C_{max}','=',num2str(OF(4))];
title ( [maintitle,' ',subtitle]);
saveas(gcf,strcat('./img/',prefix,'.png'));
```

3.2 Visualisation de frontières de dominance de Pareto (plotFront.m)

```
function plotFront(njob,nmachine)
  % read input lines
  prefix=strcat('results',num2str(njob),'x',num2str(nmachine),'_ParetoFront');
  f=fopen(strcat('./output/',prefix,'.out'));
  tline = fgetl(f);
  tlines = cell(0,1);
  nb_points =0;
  while ischar(tline)
      tlines{end+1,1} = tline;
      tline = fgetl(f);
      nb_points++;
  end
  fclose(f);
  figure;
  colors = lines(16); % no more than 16 fronts
  data = zeros(nb_points,4);
  for i=1:nb_points
    data(i,:) = sscanf(tlines{i}, '%f %f %f %f');
  endfor
  %plot objective vector
  scatter3 (data(:,2),data(:,3),data(:,4), 50, colors(data(:,1)),'filled')
  text(data(:,2)+0.1,data(:,3)+0.1,data(:,4)+0.1,num2str(data(:,1)));
  %labels and title
  xlabel("W_{tot}");
  ylabel("W_{max}");
  zlabel("C_{max}");
  title(["Frontières de Pareto pour le problem ",num2str(njob)," x ", num2str(
      nmachine)]);
endfunction
```

3.3 Visualisation des ordonnancements

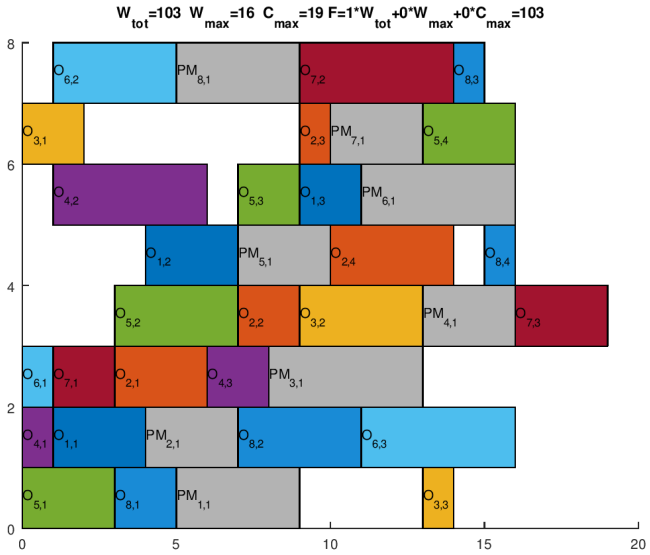

FIGURE 1 – 8×8 W_t : $(W_t, W_{max}, C_{max}) = (103; 16; 19)$

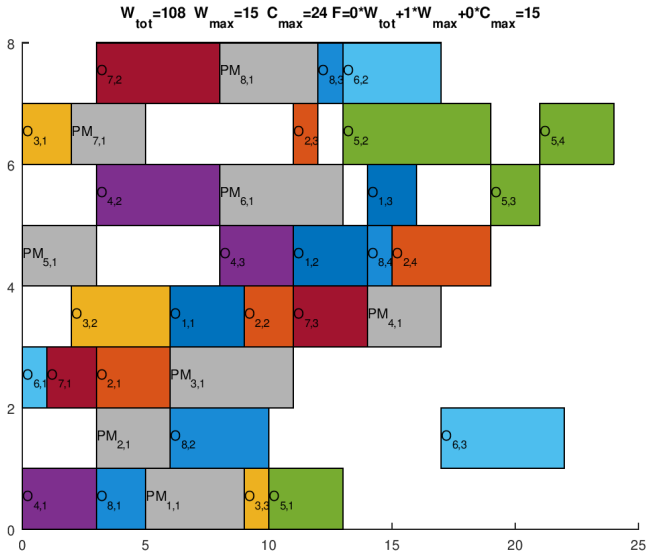

FIGURE 2 – 8×8 W_{max} : $(W_t, W_{max}, C_{max}) = (108; 15; 24)$

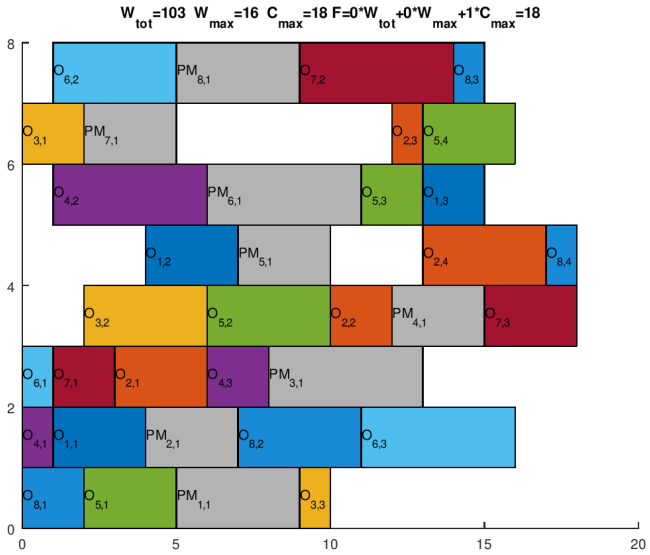

FIGURE 3 – 8×8 C_{max} : $(W_t, W_{max}, C_{max}) = (103; 16; 18)$

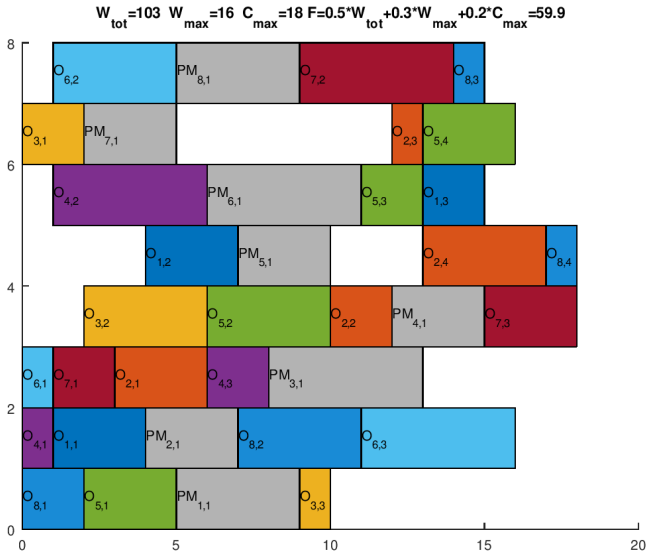

FIGURE 4 – 8×8 $F(0.5; 0.3; 0.2)$: $(W_t, W_{max}, C_{max}) = (103; 16; 18)$

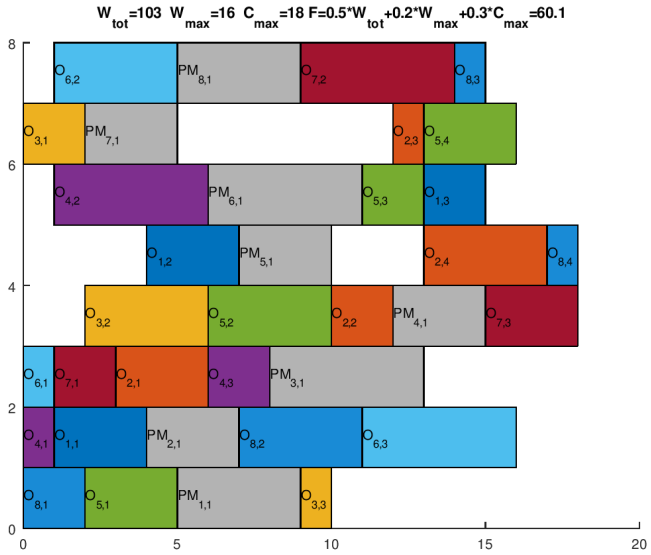

FIGURE 5 – 8×8 $F(0.5; 0.2; 0.3)$: $(W_t, W_{max}, C_{max}) = (103; 16; 18)$

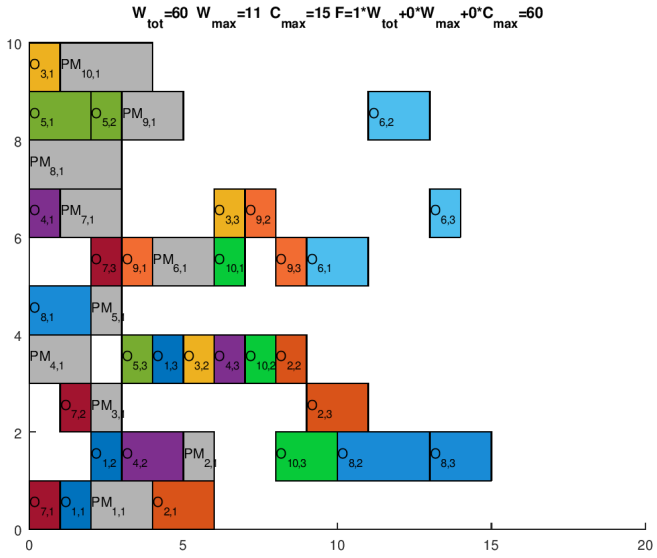

FIGURE 6 – 10×10 W_t : $(W_t, W_{max}, C_{max}) = (60; 11; 15)$

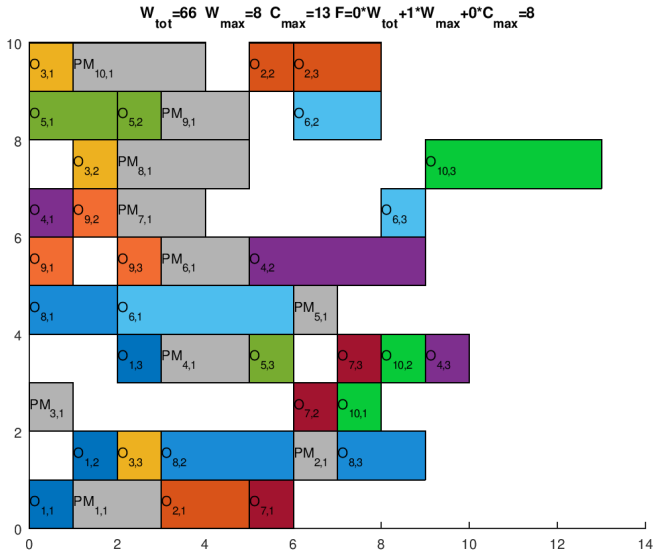

FIGURE 7 – 10×10 W_{max} : $(W_t, W_{max}, C_{max}) = (66; 8; 13)$

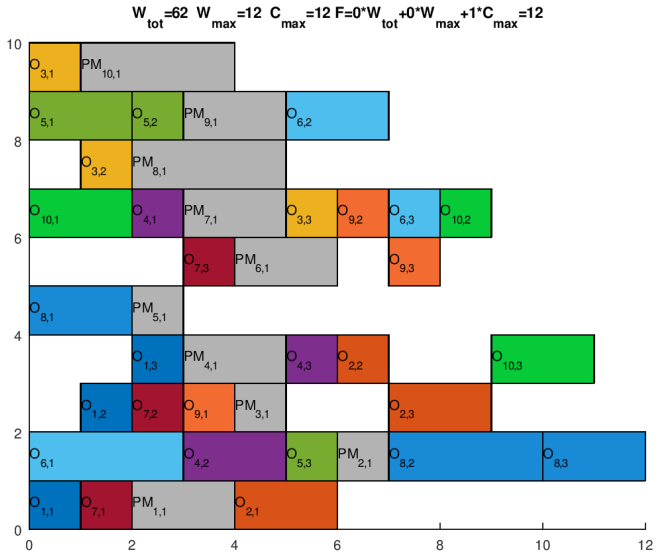

FIGURE 8 – 10×10 C_{max} : $(W_t, W_{max}, C_{max}) = (62; 12; 12)$

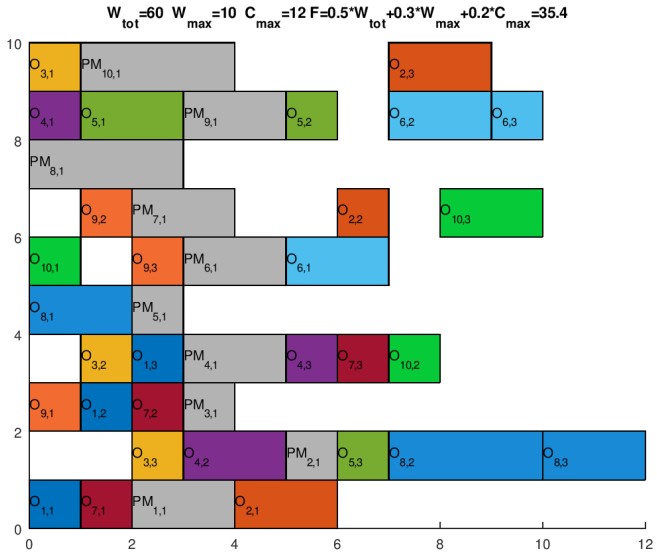

FIGURE 9 – 10×10 $F(0.5; 0.3; 0.2) : (W_t, W_{max}, C_{max}) = (60; 10; 12)$

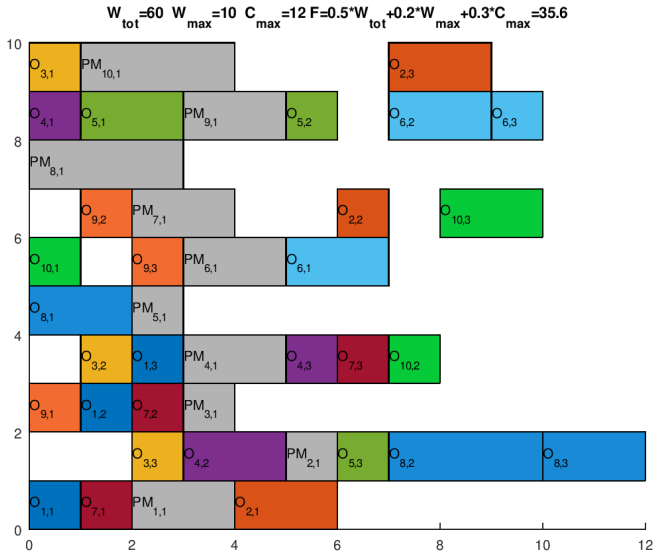

FIGURE 10 – 10×10 $F(0.5; 0.2; 0.3) : (W_t, W_{max}, C_{max}) = (60; 10; 12)$

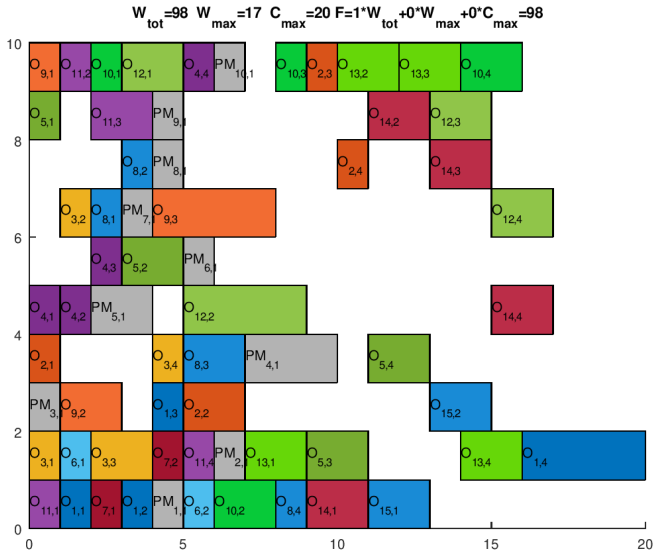

FIGURE 11 – 15×10 W_t : $(W_t, W_{max}, C_{max}) = (98; 17; 20)$

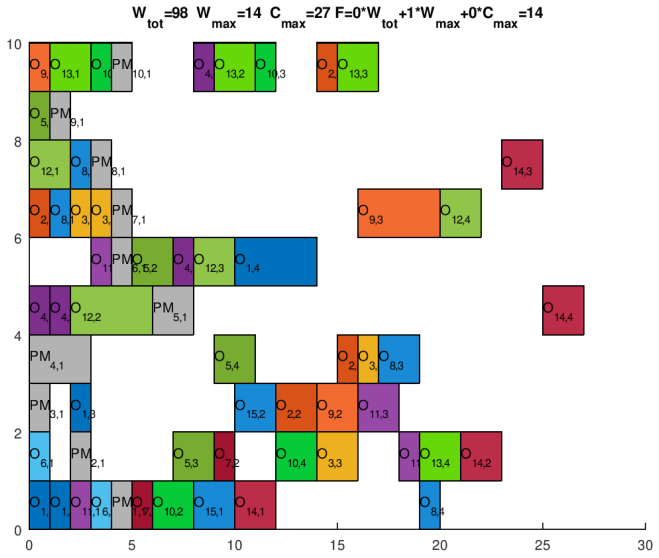

FIGURE 12 – 15×10 W_{max} : $(W_t, W_{max}, C_{max}) = (98; 14; 27)$

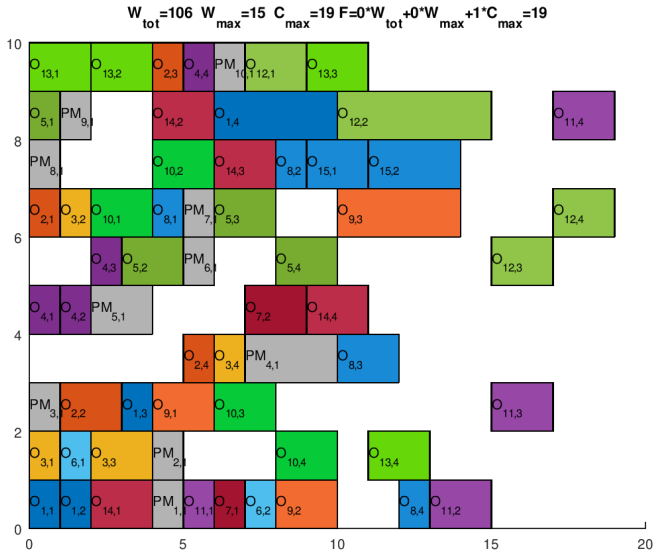

FIGURE 13 – 15×10 C_{max} : $(W_t, W_{max}, C_{max}) = (106; 15; 19)$

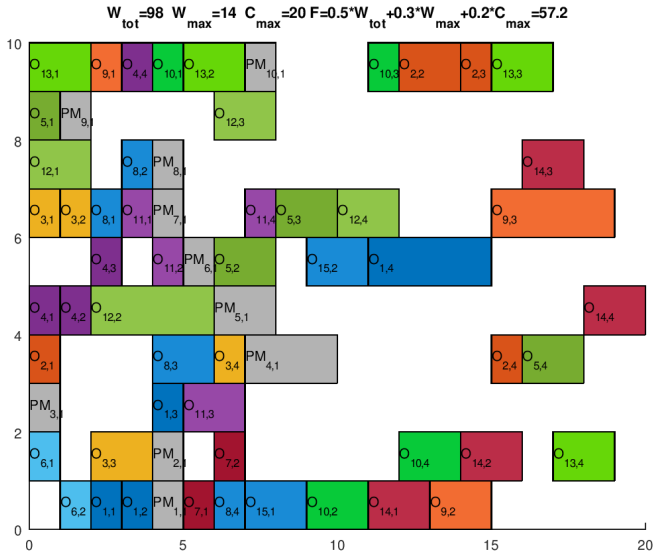

FIGURE 14 – 15×10 $F(0.5; 0.3; 0.2)$: $(W_t, W_{max}, C_{max}) = (98; 14; 20)$

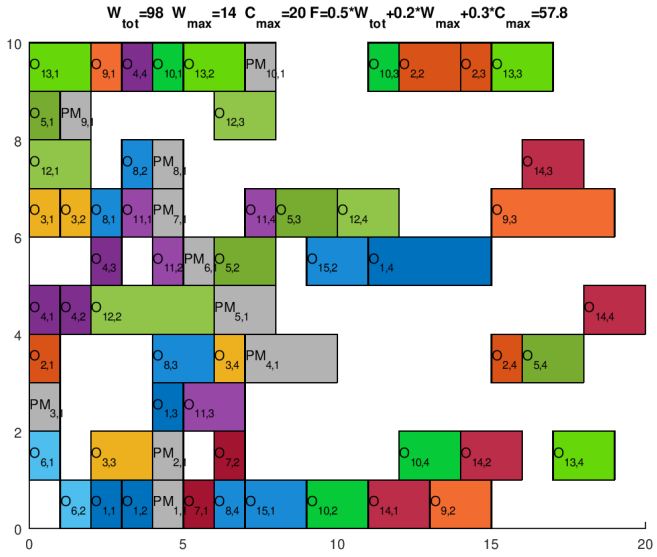

FIGURE 15 – 15×10 $F(0.5; 0.2; 0.3)$: $(W_t, W_{max}, C_{max}) = (98; 14; 20)$

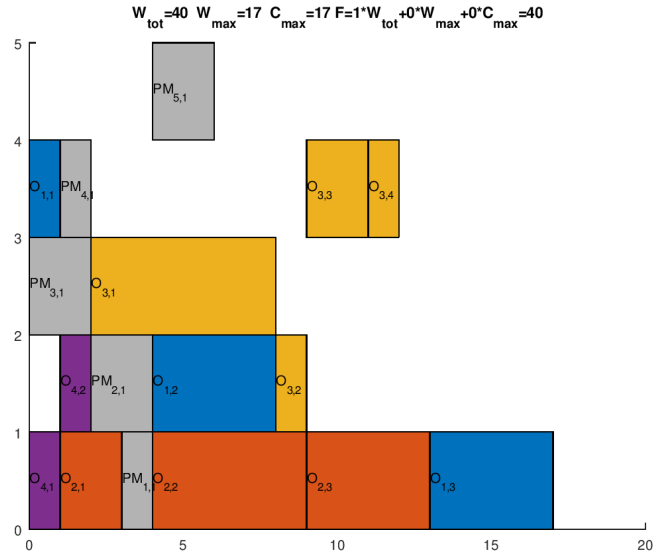

FIGURE 16 – 4×5 W_t : $(W_t, W_{max}, C_{max}) = (40; 17; 17)$

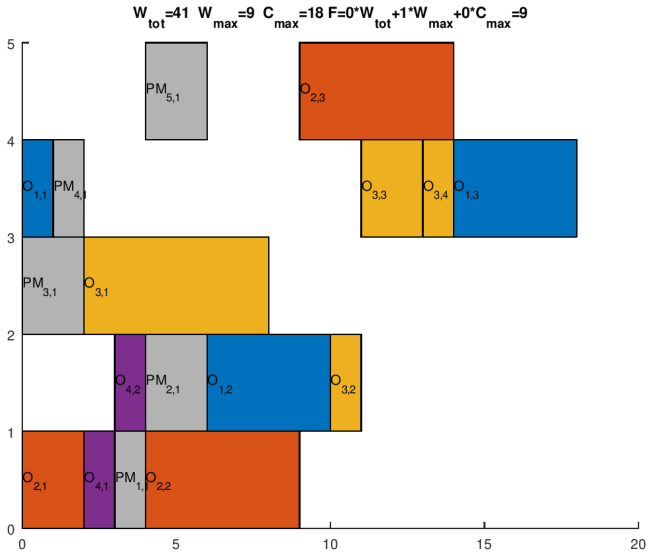

FIGURE 17 – 4×5 W_{max} : $(W_t, W_{max}, C_{max}) = (41; 9; 18)$

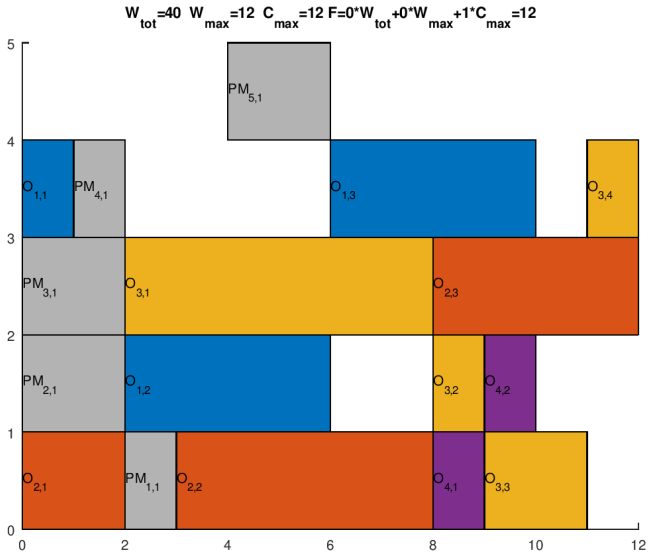

FIGURE 18 – 4×5 C_{max} : $(W_t, W_{max}, C_{max}) = (41; 12; 12)$

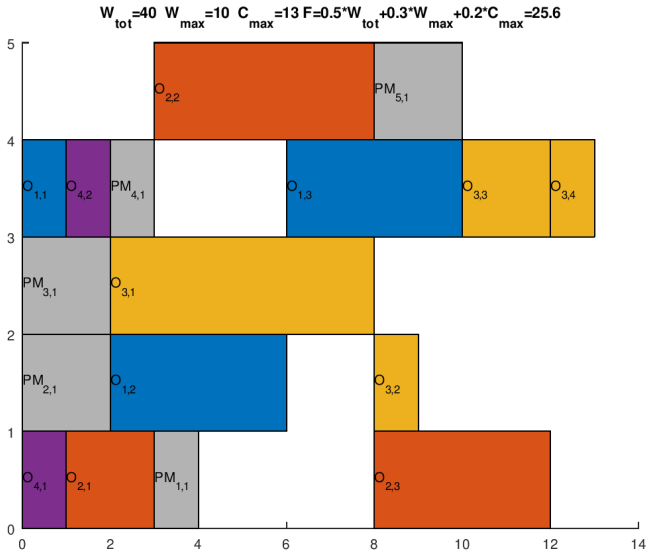

FIGURE 19 – 4×5 $F(0.5; 0.3; 0.2) : (W_t, W_{max}, C_{max}) = (40; 10; 13)$

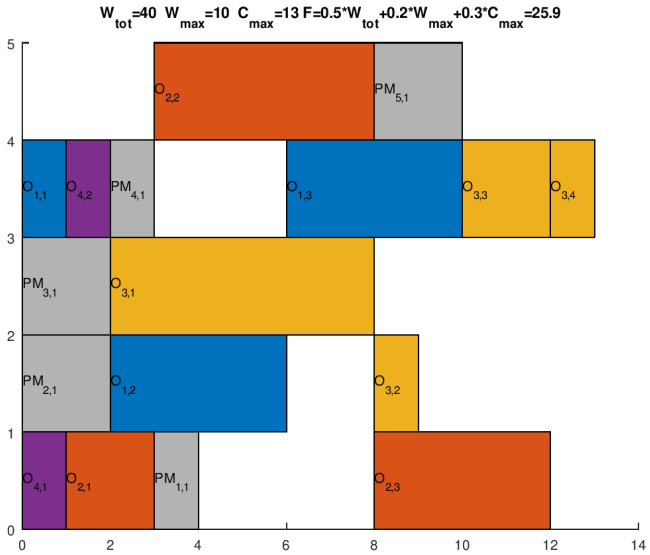

FIGURE 20 – 4×5 $F(0.5; 0.2; 0.3) : (W_t, W_{max}, C_{max}) = (40; 10; 13)$

3.4 Visualisation des frontières de dominance de Pareto

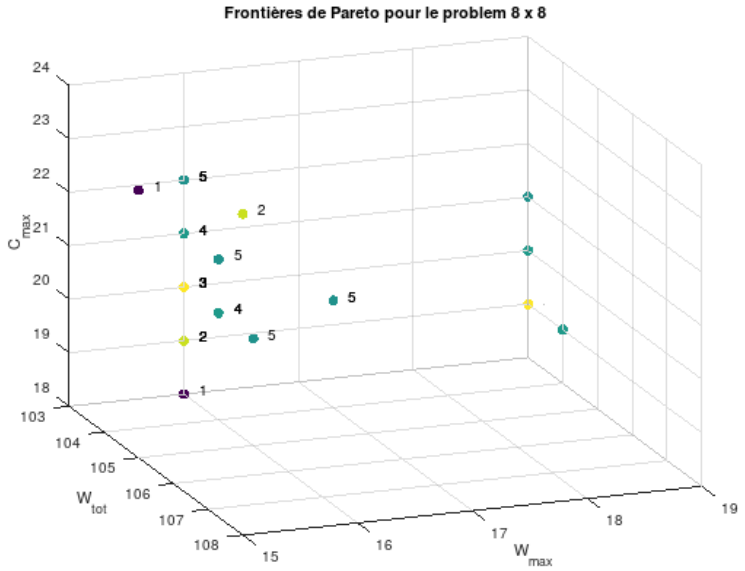

FIGURE 21 – 5 Frontières de dominance de Pareto pour le problème 8x8

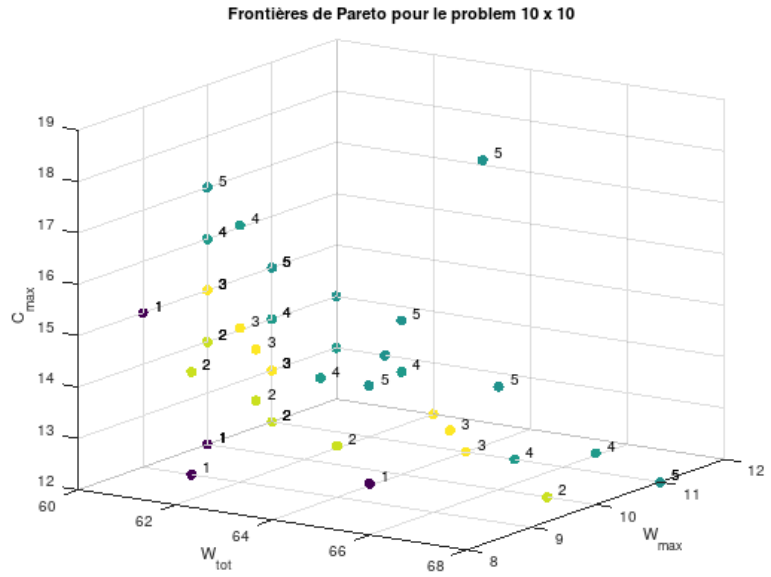

FIGURE 22 – 5 Frontières de dominance de Pareto pour le problème 10x10

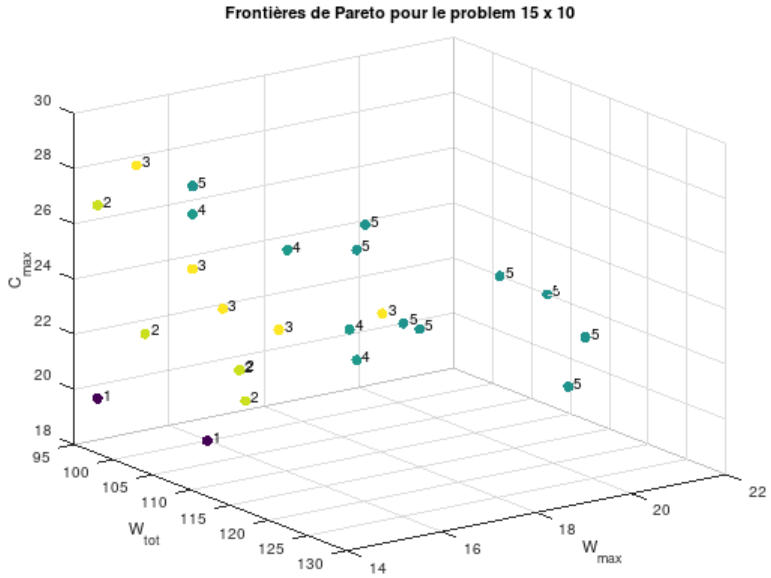

FIGURE 23 – 5 Frontières de dominance de Pareto pour le problème 15x10

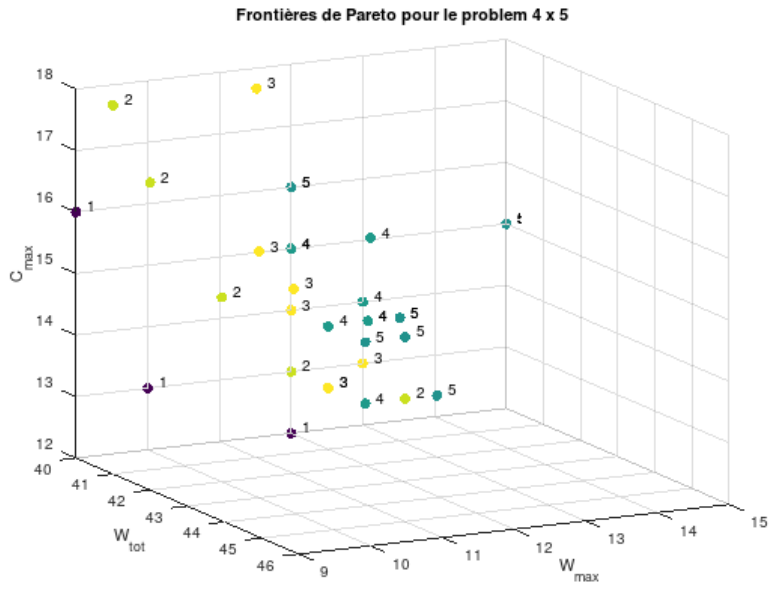

FIGURE 24 – 5 Frontières de dominance de Pareto pour le problème 4x5

Références

[1] Rajkumar, Muthukannan & Asokan, P & Vamsikrishna, V. (2010), *A GRASP algorithm for flexible job-shop scheduling with maintenance constraints*, International Journal of Production Research - INT J PROD RES. 48. 6821-6836. 10.1080/00207540903308969.

[2] F. Pezzella, G. Morganti, G. Ciaschetti, *A genetic algorithm for the Flexible Job-shop Scheduling Problem*, Computers & Operations Research, Volume 35, Issue 10, 2008, Pages 3202-3212, ISSN 0305-0548, https ://doi.org/10.1016/j.cor.2007.02.014. (http ://www.sciencedirect.com/science/article/pii/S0305054807000524)

[3] T. S. Chan, F & Wong, T. C. & Chan, LY. (2006). *Flexible job-shop scheduling problem under resource constraints*, International Journal of Production Research - INT J PROD RES. 44. 2071-2089. 10.1080/00207540500386012.