Rapport de projet OS10

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

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

5 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 au temps de préparation d'une pour le traitement d'une opération donnée mais surtout à la performance et à l'adéquation de la machine pour exécuter une telle tâche
2. la prise en compte de tâches de maintenance à effectuer dans des fenêtres de temps données (non fixité). 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 est indisponible est ne peut pas traiter d'opération, ce qui impacte fortement les performances d'une 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 de 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 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). Enfin 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 semble presque inenvisageable [1]. Ainsi ce sont les heuristiques comme l'algorithme GRASP ([1]) et les méta-heuristiques comme les algorithmes génétiques([2],[3]) qui ont rencontré le plus de succès.

1 Introduction

Avec la notation de Graham, ce problème peut s'écrire sous la forme :

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

. On ainsi un ensemble de job i Une solution à ce problème doit indiquer les affectations de chaque opération à une machine ainsi que les les temps de complétions (ou de début) de chaque opération et de chaque tâche de maintenance (et de chaque job et chaque machine). En eff

1. d'après [2], les méthodes exactes par graphes disjonctifs sont actuellement limités à 20 jobs et

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

```
   // Initialisation
1  t le nombre de solutions;
2  A = {α_1, α_2, ..., α_t} l'ensemble des valeurs α discrètes aléatoires telle que
     0 ≤ α_i ≤ 1 ∀i = 1,t;
3  S = {s_1, s_2, ..., s_t} l'ensemble des solutions à construire;
   // Assignment
4  pour chaque α_i ∈ A faire
5  |  σ l'ensemble des opérations non assignées;
6  |  s_i la solution à construire;
7  |  s_i ← ∅;
8  |  tant que σ ≠ ∅ faire
9  |  |  Chercher o_M l'opération dont le processing time est le plus longue;
10 |  |  Chercher o_m l'opréation dont le processing time est le plus court;
11 |  |  Range ← p_{o_M} − p_{o_m};
12 |  |  Width ← Range ∗ α_i;
13 |  |  Construire RCL l'ensemble des opérations non assignées o ∈ σ telle que
   |  |    p_{o_m} ≤ p_o ≤ p_{o_m} + Width;
14 |  |  Ordonner RCL dans l'ordre croissant de processing time;
15 |  |  Pondérer les éléments de RCL par la fonction linéaire 1/r où r est l'ordre
   |  |    de l'élément;
16 |  |  Choisir de manière aléatoire un élément o* de RCL en tenant compte
   |  |    des pois ci-dessus;
17 |  |  s_i ← s_i ∪ o*;
18 |  |  σ ← σ\o*;
19 |  fin
20 |  Calculer la charge totale de s_i;
21 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**
  - Chercher o_M l'opération dont le processing time est le plus longue ; et o_m l'opréation dont le processing time est le plus court en tenant compte les maintenances qui doivent être situées au milieu des intervalles donnés $\left[t_{jl}^E, t_{jl}^L\right]$;
  - $Range \leftarrow p_{o_M} - p_{o_m}$;
  - $Width \leftarrow 1 + Range * \alpha_i$;
  - Construire RCL l'ensemble des opérations non assignées $o \in \sigma$ telle que $p_{o_m} \leq p_o \leq p_{o_m} + Width$;
  - Ordonner RCL dans l'ordre croissant de processing time;
  - Pondérer les éléments de RCL par la fonction linéaire $\frac{1}{r}$ où r est l'ordre de l'élément;
  - Choisir de manière aléatoire un élément o^* de RCL en tenant compte des pois ci-dessus;
  - Ordonner o^* en tenant compte la date de disponibilité la plus tôte et les maintenances;
  - $\sigma \leftarrow \sigma \backslash o^*$;
- **fin**
- Ordonner les maintenances au milieu des intervalles donnés $\left[t_{jl}^E, t_{jl}^L\right]$ sans collision avec les opérations scédulées;
- Calculer le makespan 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	13
$F(0.5; 0.3; 0.2)$	59.9	60.1
$F(0.5; 0.2; 0.3)$	60.1	60.4

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	11
$F(0.5; 0.3; 0.2)$	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	20
$F(0.5; 0.3; 0.2)$	60.3	57.2
$F(0.5; 0.2; 0.3)$	60.9	57.8

TABLE 3 – Résultat pour problème 15×10

10 machines pour le FJSSP (sans maintenance)

Critères	Valeurs de l'article	Nos valeurs
W_t	40	40
W_{max}	9	9
C_{max}	16	13
$F(0.5; 0.3; 0.2)$	25.9	26
$F(0.5; 0.2; 0.3)$	26.6	26.5

TABLE 4 – Résultat pour problème 4×5

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.

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 (3.4).

3 Annexes

3.1 Visualisation de tournées (plots.m)

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

% read input
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');
OFVS =  sscanf(tlines{3}, '%f %f %f');
data_jobs = zeros(n(1), 5);
```

```
data_maint = zeros(n(2), 4);
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));

%text(starts(n,1)+0.1,starts(n,2)+0.1, num2str(number(n)));
figure;
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);
  color = mod(color,1);
  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
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,'\n',subtitle]);
saveas(gcf,strcat('./img/',prefix,'.png'));
```

3.2 Automatisation de traçage des exemplaires (tests.m)

```
types={'Cmax','Wmax','Wtot','F050203','F050302'};
vals=[4 5;8 8;10 10;15 10];
for i=1:length(vals)
    for j=1:length(types)
        plotJob(vals(i,1),vals(i,2),types{j});
    end
end
```

3.3 Visualisation de statistiques (stats.R)

3.4 Visualisation des ordonnacements

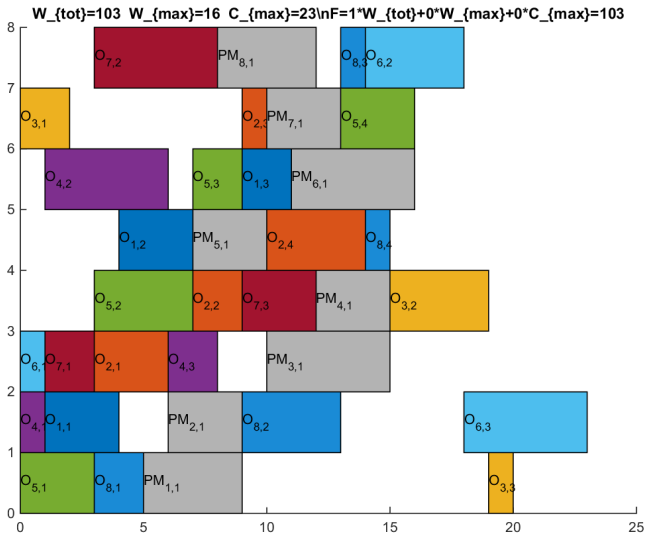

FIGURE 1 – 8×8 W_t : $(W_t, W_{max}, C_{max}) = (40; 17; 17)$

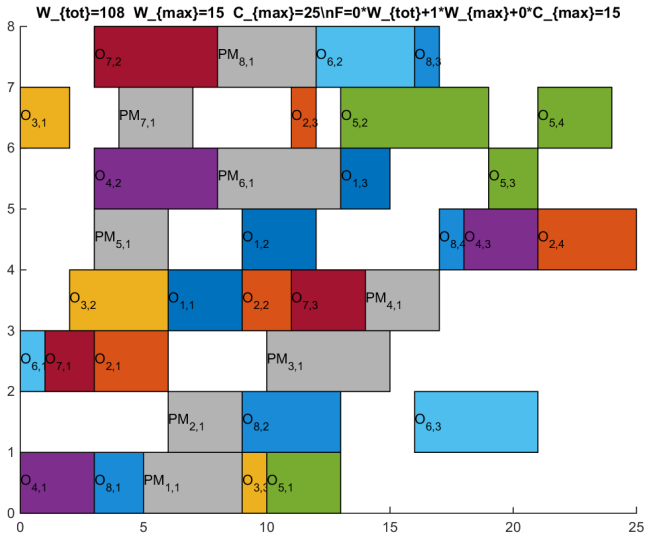

FIGURE 2 – 8×8 W_{max} : $(W_t, W_{max}, C_{max}) = (41; 9; 18)$

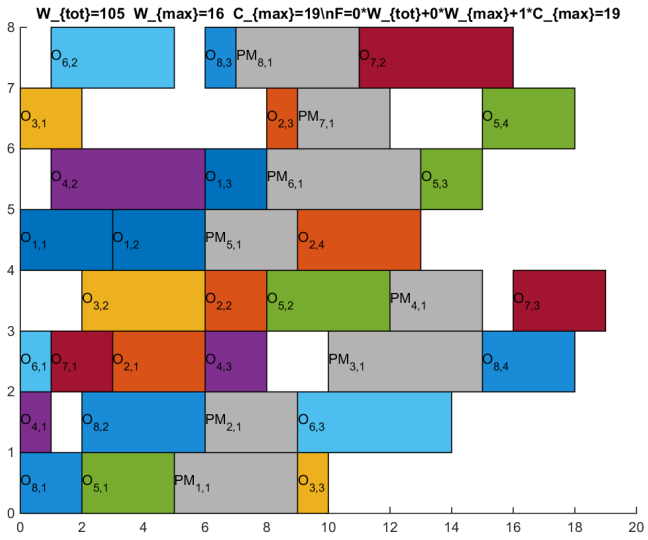

FIGURE 3 – 8×8 C_{max} : $(W_t, W_{max}, C_{max}) = (105; 16; 19)$

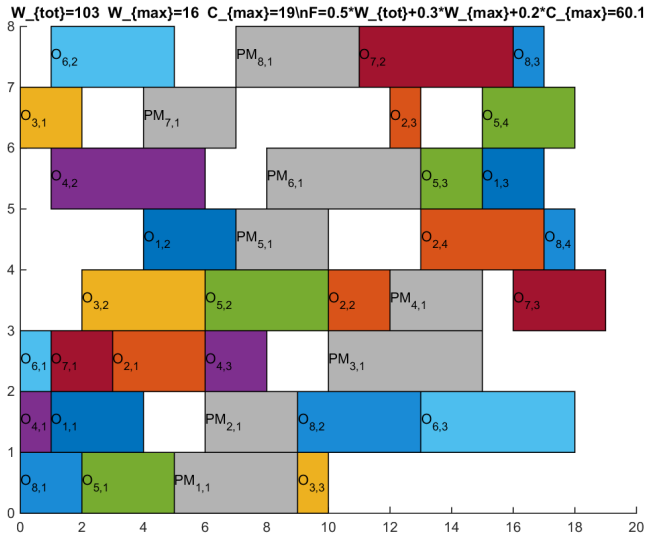

FIGURE 4 – 8×8 $F(0.5; 0.3; 0.2)$: $(W_t, W_{max}, C_{max}) = (40; 10; 15)$

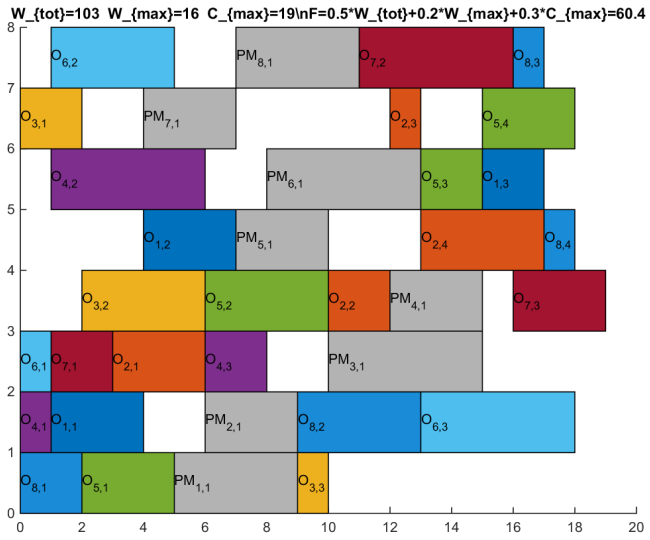

FIGURE 5 – 8×8 $F(0.5; 0.2; 0.3)$: $(W_t, W_{max}, C_{max}) = (40; 10; 15)$

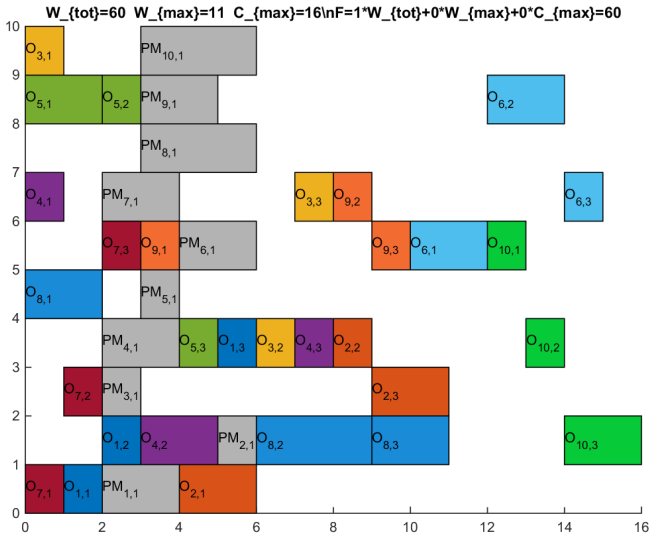

FIGURE 6 – 10×10 W_t : $(W_t, W_{max}, C_{max}) = (60; 11; 16)$

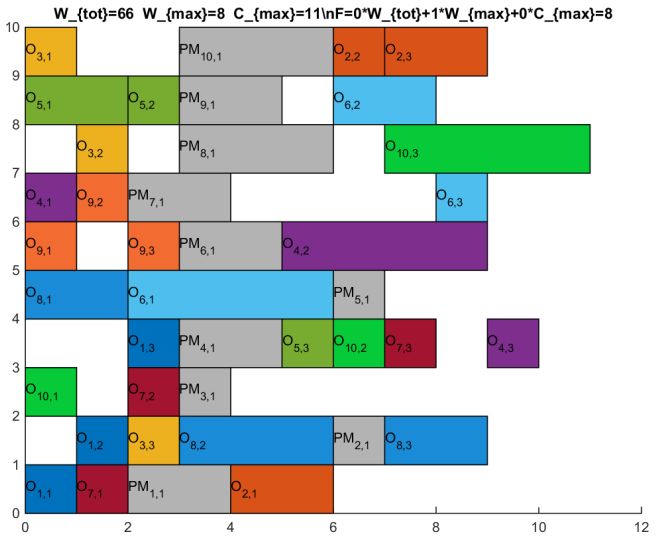

FIGURE 7 – 10×10 W_{max} : $(W_t, W_{max}, C_{max}) = (66; 8; 11)$

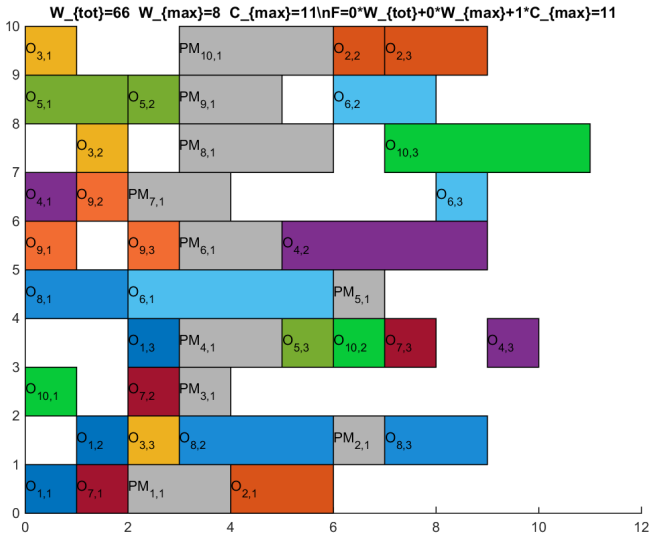

FIGURE 8 – 10×10 C_{max} : $(W_t, W_{max}, C_{max}) = (66; 8; 11)$

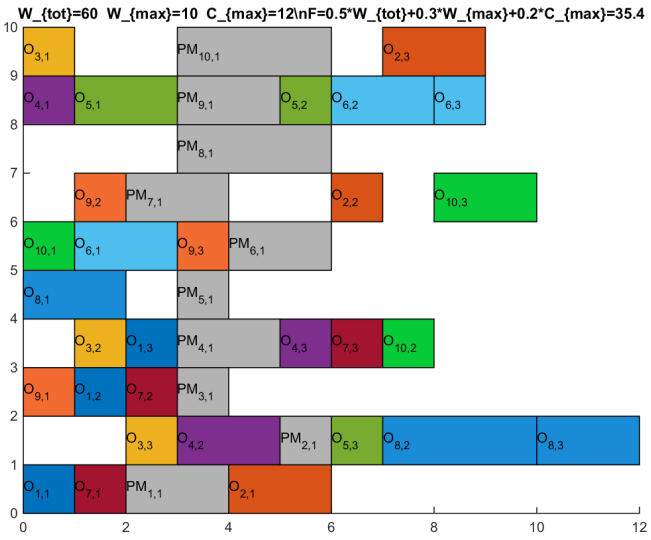

FIGURE 9 – 10×10 $F(0.5; 0.3; 0.2)$: $(W_t, W_{max}, C_{max}) = (60; 10; 12)$

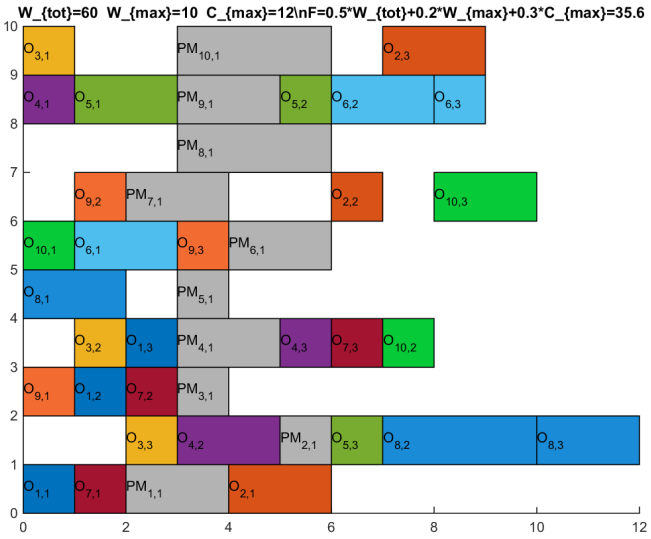

FIGURE 10 – 10×10 $F(0.5; 0.2; 0.3)$: $(W_t, W_{max}, C_{max}) = (60; 10; 12)$

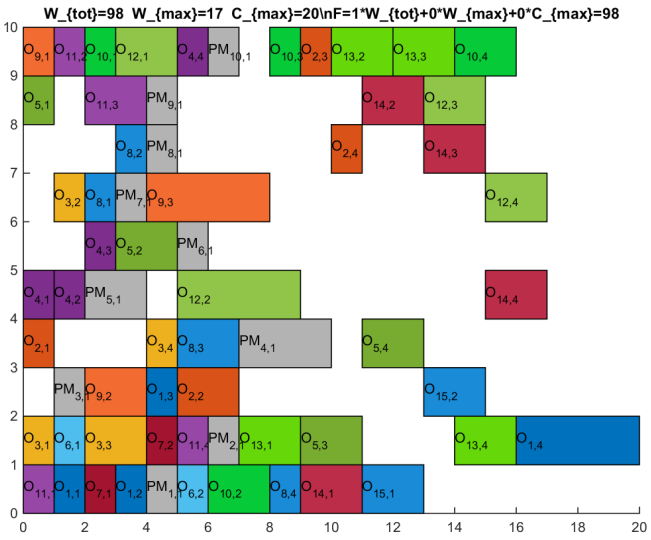

FIGURE 11 – 15×10 W_t : $(W_t, W_{max}, C_{max}) = (98; 17; 20)$

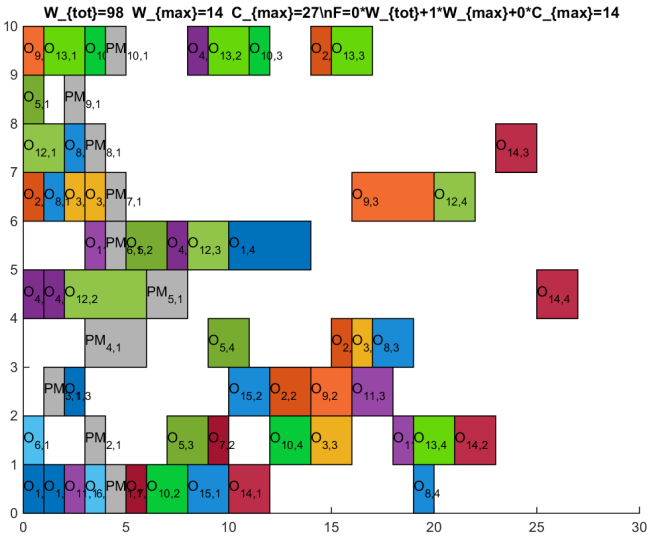

FIGURE 12 – 15×10 W_{max} : $(W_t, W_{max}, C_{max}) = (98; 14; 27)$

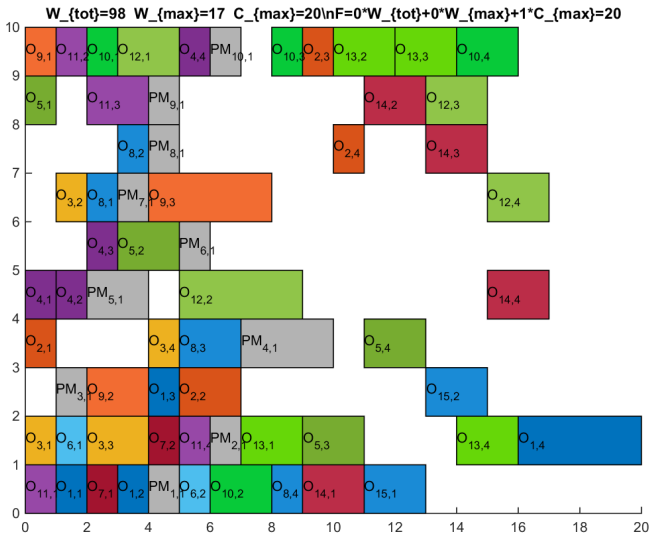

FIGURE 13 – 15×10 C_{max} : $(W_t, W_{max}, C_{max}) = (98; 17; 20)$

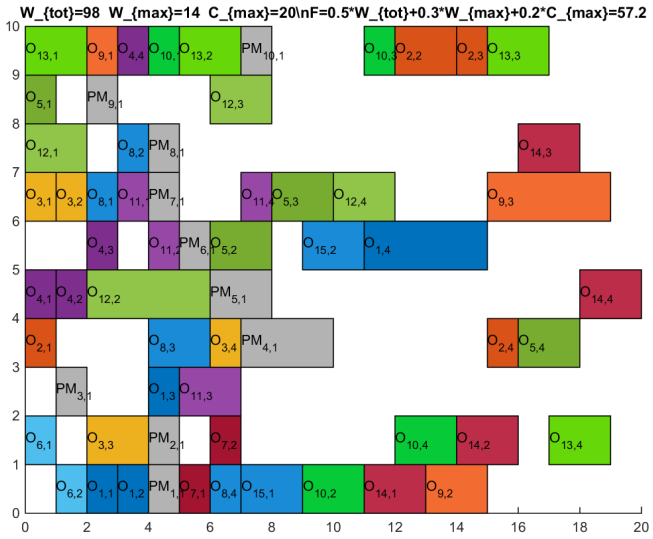

FIGURE 14 – 15×10 $F(0.5; 0.3; 0.2)$: $(W_t, W_{max}, C_{max}) = (98; 14; 20)$

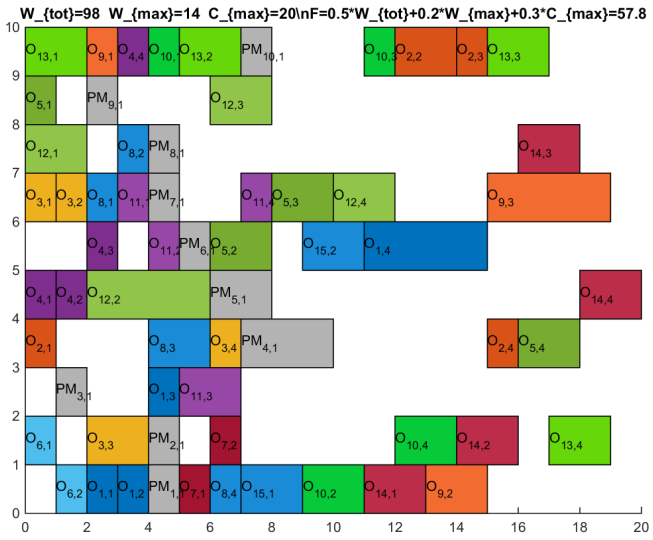

FIGURE 15 – 15×10 $F(0.5; 0.2; 0.3)$: $(W_t, W_{max}, C_{max}) = (98; 14; 20)$

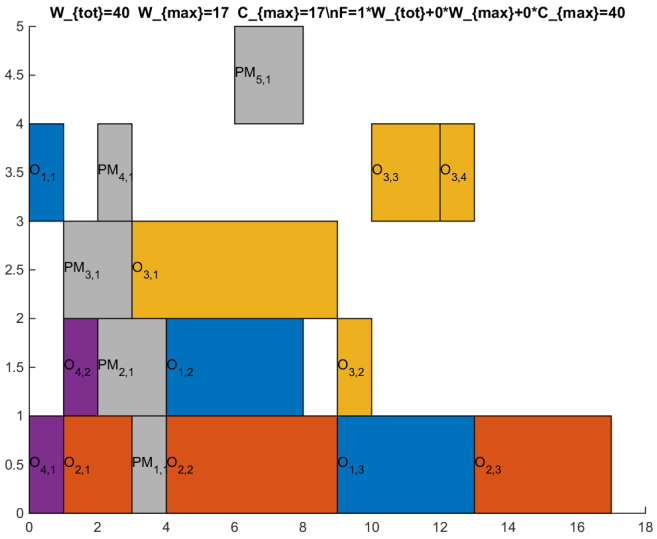

FIGURE 16 – 4×5 W_t : $(W_t, W_{max}, C_{max}) = (40; 17; 17)$

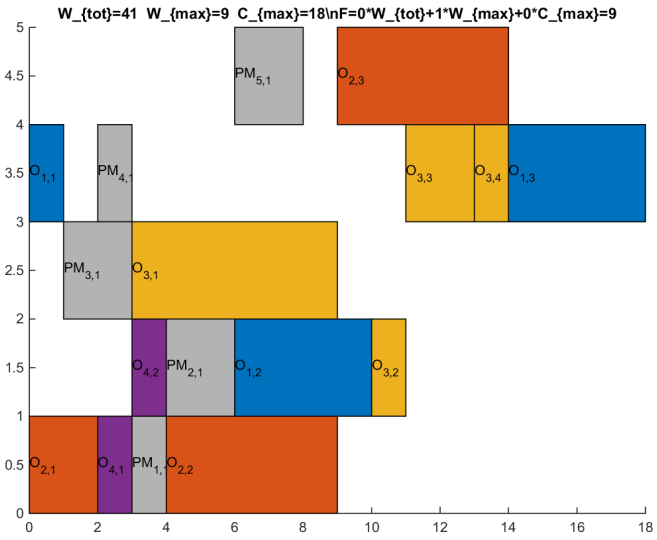

FIGURE 17 – 4×5 W_{max} : $(W_t, W_{max}, C_{max}) = (41; 9; 18)$

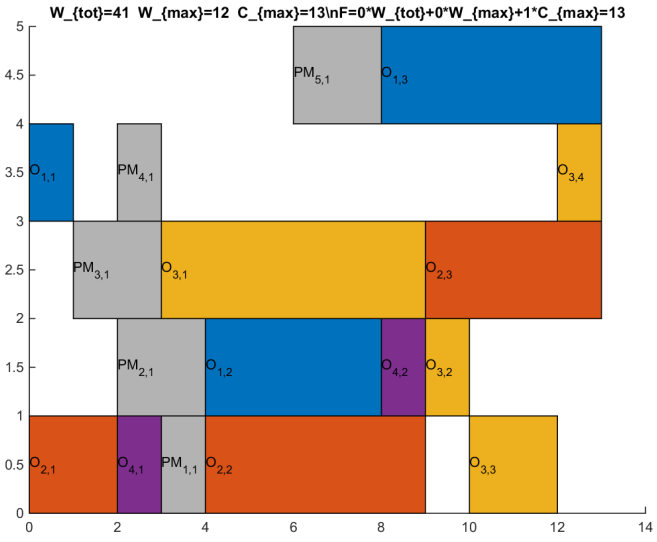

FIGURE 18 – 4×5 C_{max} : $(W_t, W_{max}, C_{max}) = (41; 12; 13)$

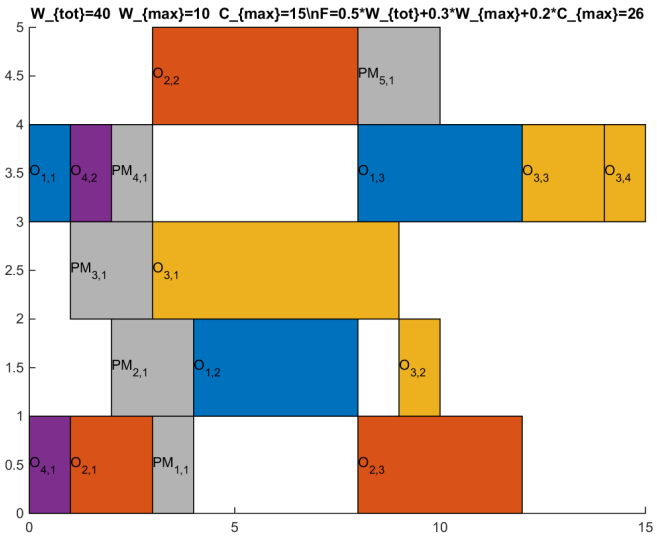

FIGURE 19 – 4×5 $F(0.5; 0.3; 0.2)$: $(W_t, W_{max}, C_{max}) = (40; 10; 15)$

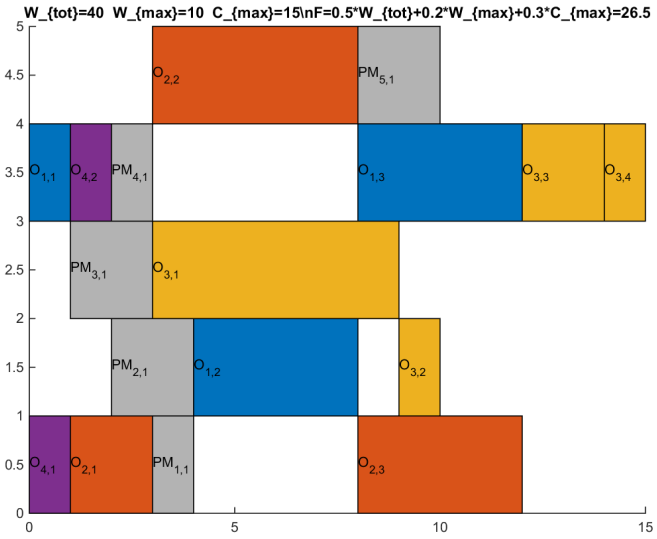

FIGURE 20 – 4×5 $F(0.5; 0.2; 0.3)$: $(W_t, W_{max}, C_{max}) = (40; 10; 15)$

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.