

CHAPITRE III

Application de la méthode tabou sur le problème d'ordonnancement d'atelier

3.1. Introduction

Dans un chapitre on veut appliquer la méthode de la recherche tabou pour résoudre le problème P^* .

3.2. Définition du problème P^*

On considère le problème d'ordonnancement de n tâches sur une machine unique qui peut exécuter qu'une seule tâche à la fois. Une tâche i est caractérisée par la donnée d'un temps d'exécution p_i , un poids w_i liée à son importance et une date de disponibilité r_i . L'objectif est de déterminer un ordonnancement de ces tâches sur cette machine.

Afin de minimiser la somme pondérée des dates de fin.

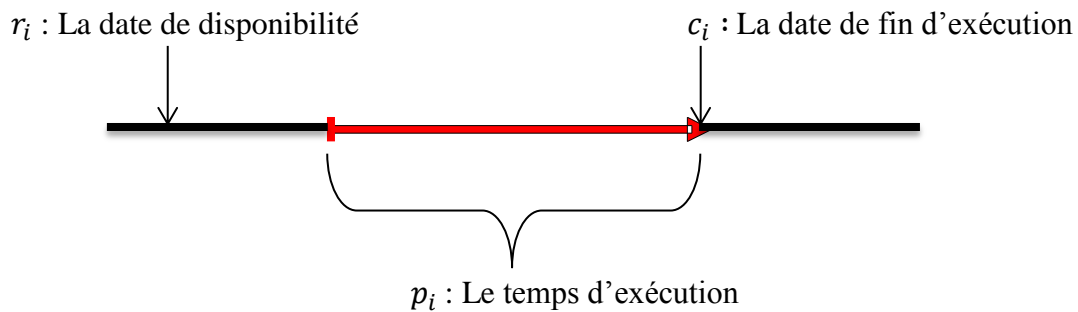


Figure (3.1) : Caractéristique d'une tâche du problème P^* .

3.3. L'historique de problème

Ce problème est connu d'être NP-difficile comme la plupart des problèmes d'ordonnancement [PON09]. Cette classe de problèmes se distingue par une croissance rapide du nombre de solutions possibles avec une croissance modeste du nombre de ressources pour être programmées. La croissance est si rapide que même l'ordinateur le plus rapide ne pourrait pas chercher à travers chaque solution potentielle aux problèmes à grande échelle dans un laps

de temps raisonnable, il est difficile de saisir la formulation du problème dans une expression mathématique. Ceci est classé pour leurs problèmes d'ordonnement au sein de NP-difficile. Donc un algorithme polynomial pour déterminer une solution exacte est impossible.

Et par conséquent pour résoudre un problème NP-Difficile on doit recourir à l'utilisation des méthodes de solution approchée plutôt qu'exactes.

Pour le cas où la taille est assez grand la méthode de Branch and Bound et la programmation dynamique échouent pour déterminer dans un temps raisonnable une solution exacte. Par conséquent nous nous trouvons dans l'obligation d'utiliser d'autre méthode plus rapide, même si elle détermine une solution approchée parmi ces méthodes connue sous le nom méthahuristique on utilise la méthode tabou. Une méthode prouvant sa supériorité relativement aux autres méthodes approchées lors de son application pour résoudre quelques problèmes d'optimisation combinatoire.

3.4. Les voisinages :

3.4.1. Le voisinage par insertion de tâches V_I :

Le voisinage par insertion est un algorithme de tri classique dans l'informatique et généralement considéré comme le tri le plus efficace sur des entrées de petite taille. Il est aussi très rapide lorsque les données sont déjà presque triées. Pour ces raisons, il est utilisé en pratique en combinaison avec d'autres méthodes comme le tri rapide.

Ce type de voisinage est le choix de deux tâches différentes de la séquence en cours est la première tâche insérer dans le deuxième tâche place puis la troisième à la dernière progressivement. Nous générons $n(n-1)/2$ séquence de la séquence d'origine.

Exemple :

$T = (2, 3, 1, 4, 5)$.

Itération = 1 $(2, 3, 1, 4, 5) \longrightarrow (3, 2, 1, 4, 5)$

Itération = 2 $(3, 2, 1, 4, 5) \longrightarrow (3, 1, 2, 4, 5)$

Itération = 3 $(3, 1, 2, 4, 5) \longrightarrow (3, 1, 4, 2, 5)$

Itération = 4 $(3, 1, 4, 2, 5) \longrightarrow (3, 1, 4, 5, 2)$

Et comme suit avec 3, 1, 4, 5 pour trouver $|V(T)| = 10$.

3.4.2. Le voisinage par permutation de deux tâches V_p :

Ce type de voisinage est basé sur les deux stratégies :

- **La permutation simple de deux tâches adjacentes :**

Cette stratégie dépend de l'interrupteur et tourner chaque paire de tâches adjacentes

Nous générons $(n - 1)$ séquence de la séquence d'origine.

Exemple :

On prend le même exemple précédent $T = (2, 3, 1, 4, 5)$.

$$V(T) = \{ (3, 2, 1, 4, 5), (2, 1, 3, 4, 5), (2, 3, 4, 1, 5), (2, 3, 1, 5, 4) \}.$$

- **La permutation simple de deux tâches :**

Cette stratégie repose sur la conversion de chaque paire de tâches (pas nécessairement adjacentes).

Et nous obtenons grâce à eux $n(n - 1)/2$ séquence de la séquence d'origine.

Exemple :

$$|V(T)| = 10.$$

$$T = (2, 3, 1, 4, 5).$$

$$V(T) = \{ (3, 2, 1, 4, 5), (1, 3, 2, 4, 5), (4, 3, 1, 2, 5), (5, 3, 1, 4, 2), \dots \}.$$

3.5. Exemple illustrative :

Soit le problème de 5 tâches, avec une seule machine. On fixe la taille de liste tabou à 3, avec le nombre d'itération est 6.

i	1	2	3	4	5
p_i	3	2	6	9	7
w_i	4	8	5	3	2
r_i	9	13	10	8	15

La séquence initiale générée aléatoirement est $S_0 = (2, 3, 1, 4, 5)$ d'où $f_0 = 500$.

On adopte la stratégie de sélection de la permutation du meilleur voisin parmi tous les voisins générés par le voisinage par insertion.

Initialisation :

$$N_{itér_{max}} = 6$$

$$S^* \leftarrow (2,3,1,4,5) \quad \text{et} \quad f^* = 500$$

$$\text{Liste_tabou} \leftarrow \emptyset$$

$$i \leftarrow 1$$

Itération 1:

Le voisinage est :

$S_1 :$	(3, 2 ,1,4,5)	(3,1, 2 ,4,5)	(3,1,4, 2 ,5)	(3,1,4,5, 2)
$f_1 :$	472	488	594	606

Meilleur voisin

Non tabou

$$S_0 \leftarrow (3,2,1,4,5)$$

$$\text{Liste_tabou} \leftarrow \{(3,2,1,4,5)\}$$

$$i \leftarrow 2$$

Itération 2 :

Le voisinage est :

$S_1 :$	(2, 3 ,1,4,5)	(2,1, 3 ,4,5)	(2,1,4, 3 ,5)	(2,1,4,5, 3)
$f_1 :$	500	491	518	541

Meilleur voisin

Non tabou

$$S_0 \leftarrow (2,1,3,4,5)$$

$$\text{Liste_tabou} \leftarrow \{(2,1,3,4,5), (3,2,1,4,5)\}$$

$$i \leftarrow 3$$

Itération 3 :

Le voisinage est :

S_1 :	(1,2,3,4,5)	(1,3,2,4,5)	(1,3,4,2,5)	(1,3,4,5,2)
f_1 :	437	457	523	505

Meilleur voisin

Non tabou

$S_0 \leftarrow (1,2,3,4,5)$

Liste_tabou $\leftarrow \{(1,2,3,4,5), (2,1,3,4,5), (3,2,1,4,5)\}$

$i \leftarrow 4$

Itération 4 :

Le voisinage est :

S_1 :	(2,1,3,4,5)	(2,3,1,4,5)	(2,3,4,1,5)	(2,3,4,5,1)
f_1 :	491	500	536	555

Meilleur voisin

Meilleur voisin

mais tabou

non tabou

$S_0 \leftarrow (2,3,1,4,5)$

Liste_tabou $\leftarrow \{(2,3,1,4,5), (1,2,3,4,5), (2,1,3,4,5)\}$

$i \leftarrow 5$

Itération 5 :

Le voisinage est :

S_1 :	(3,2,1,4,5)	(3,1,2,4,5)	(3,1,4,2,5)	(3,1,4,5,2)
f_1 :	472	488	594	606

Meilleur voisin

Non tabou

$S_0 \leftarrow (3,2,1,4,5)$

Liste_tabou $\leftarrow \{(3,2,1,4,5), (2,3,1,4,5), (1,2,3,4,5)\}$

$i \leftarrow 6$

Itération 6 :

Le voisinage est :

S_1 :	(2,3,1,4,5)	(2,1,3,4,5)	(2,1,4,3,5)	(2,1,4,5,3)
f_1 :	500	491	518	541

Meilleur voisin

Non tabou

$S_0 \leftarrow (2,1,3,4,5)$

Mise à jour de liste tabou

Liste_tabou $\leftarrow \{(2,1,3,4,5), (3,2,1,4,5), (2,3,1,4,5)\}$

$i \leftarrow 7 \dots \text{stop.}$

Le processeur de recherche s'arrête et la meilleure solution trouvée :

$S^* = (1,2,3,4,5)$ avec $\sum_{i=1}^5 w_i C_i = 437$.

3.6. Le Choix des paramètres de la méthode

Les voisinages qui ont été sélectionnés pour la recherche tabou, on adopte donc les deux types de voisinage précédent : génération par permutation de deux tâches et le voisinage par insertion.

Des recherches récentes ont prouvé que le meilleur choix de la taille de la liste tabou est \sqrt{n} où n la taille des problèmes considéré.

Le critère d'arrêt de la méthode est lorsqu'on atteint un certain nombre d'itérations dite maximale $N_{it\acute{e}r}$.

Dans nos tests on a considéré deux cas $N_{it\acute{e}r} = 20$ et $N_{it\acute{e}r} = 40$ et ceci pour éviter le blocage du processeur de la recherche d'une part et d'autre part pour faire progresser l'exécution de la méthode.

3.7. Test des résultats

Nous avons essayé quelques expériences et l'exécution du programme et les résultats obtenus ont été classés dans les Tables.

Lorsque le premier tableau représente les résultats du voisinage par insertion et deuxième tableau représente les résultats du voisinage par permutation.

La taille	V_I											
	Les problèmes	$N_{itér} = 20$					$N_{itér} = 40$					
		Le temps d'exécution	La fonction objectif			Le temps d'exécution	La fonction objectif					
		F_0	F_f	Ra F		F_0	F_f	Ra F		F_0	F_f	Ra F
10	P_1	0.300805	30052	1817	0.060461866	0.468932	1556	1556	1			
	P_2	0.234396	21197	1323	0.062414493	0.492802	1120	1120	1			
	P_3	0.249870	34252	2260	0.065981549	0.413839	25170	1616	0.064203417			
	P_4	0.261851	11966	1891	0.158031088	0.449307	18684	1403	0.075090987			
	P_5	0.237052	29439	2538	0.086212168	0.637161	29152	2165	0.074265917			
	Moy	0.2567948	25381	1965	0.077420117	0.4924082	15136	1572	0.103858351			
100	P_1	35.032335	1216330	17317	0.01423709	54.774665	1021683	15027	0.014708085			
	P_2	25.899458	1141595	20106	0.0176122	53.870633	181521	15963	0.087940238			
	P_3	33.457694	23529	19063	0.810191678	51.540111	24318	18399	0.756600049			
	P_4	28.805123	1038999	17817	0.017148236	52.623844	20620	13944	0.676236663			
	P_5	30.293535	1417743	18963	0.013375485	51.942686	22941	20682	0.901530012			
	Moy	30.697629	967639	18653	0.019276817	52.9503878	254216	16803	0.066097335			
150	P_1	68.161136	32226	26796	0.831502513	134.818923	2556944	26764	0.010467183			
	P_2	66.510955	2707029	28561	0.010550681	139.398727	116639	29409	0.252136935			
	P_3	83.902816	299994	24967	0.083224998	115.939829	2071115	24259	0.011713014			
	P_4	64.594395	2507090	24780	0.009883969	119.112502	30766	26600	0.864590782			
	P_5	65.460640	31937	25301	0.792215925	114.350853	1997613	19760	0.009891806			
	Moy	69.7259884	1115655	26764	0.023989495	124.7241668	1354615	25358	0.01871971			

Table (3.1) : Les résultats de la méthode tabou en utilisant V_I .

V _P													
		N _{itér} = 20					N _{itér} = 40]						
La taille	Les problèmes	Le temps d'exécution	La fonction objectif			Le temps d'exécution	La fonction objectif			Le temps d'exécution			
			F ₀	F _f	Ra F		F ₀	F _f	Ra F	F ₀	F _f	Ra F	
10	P ₁	0.235595	24730	1354	0.054751314	0.445984	851	851	1	0.445984	851	851	1
	P ₂	0.264246	1036	1036	1	0.456891	14913	757	0.050761081	0.456891	14913	757	0.050761081
	P ₃	0.258410	4024	1564	0.388667992	0.432225	20767	2211	0.106466991	0.432225	20767	2211	0.106466991
	P ₄	0.280215	22642	1661	0.073359244	0.636622	19842	1390	0.070053422	0.636622	19842	1390	0.070053422
	P ₅	0.240686	33082	2925	0.088416662	0.495247	1858	1178	0.63401507	0.495247	1858	1178	0.63401507
	Moy	0.2558304	17102	1708	0.09987136	0.4933938	11646	1277	0.109651382	0.4933938	11646	1277	0.109651382
100	P ₁	28.804155	16615	16615	1	48.721451	1172427	16986	0.014487896	48.721451	1172427	16986	0.014487896
	P ₂	28.361923	1227235	22621	0.018432493	53.384822	24306	19712	0.81099317	53.384822	24306	19712	0.81099317
	P ₃	25.670320	1171878	18566	0.015842946	50.218151	981772	17653	0.017980753	50.218151	981772	17653	0.017980753
	P ₄	28.708993	1060033	18299	0.01726267	51.905644	22920	16268	0.709773124	51.905644	22920	16268	0.709773124
	P ₅	25.990838	21449	19810	0.923586181	57.061530	22045	18366	0.833114085	57.061530	22045	18366	0.833114085
	Moy	27.5072458	699442	19182	0.027424719	52.2583196	444694	17797	0.040020778	52.2583196	444694	17797	0.040020778
150	P ₁	60.809974	2554820	29291	0.011464996	119.728525	2325037	28837	0.012402813	119.728525	2325037	28837	0.012402813
	P ₂	68.338350	35128	30204	0.859826919	141.607972	36092	29505	0.817494182	141.607972	36092	29505	0.817494182
	P ₃	64.342809	2258592	23577	0.010438804	130.634426	2072644	20092	0.009693898	130.634426	2072644	20092	0.009693898
	P ₄	61.124015	2341350	25553	0.010913789	123.125031	27066	23818	0.879997044	123.125031	27066	23818	0.879997044
	P ₅	62.028163	32243	23903	0.741339205	137.529712	27619	25877	0.936927477	137.529712	27619	25877	0.936927477
	Moy	63.3286622	1444426	26505	0.01834985	130.5251332	897691	25625	0.028545457	130.5251332	897691	25625	0.028545457

Table (3.2) : Les résultats de la méthode tabou en utilisant V_p .

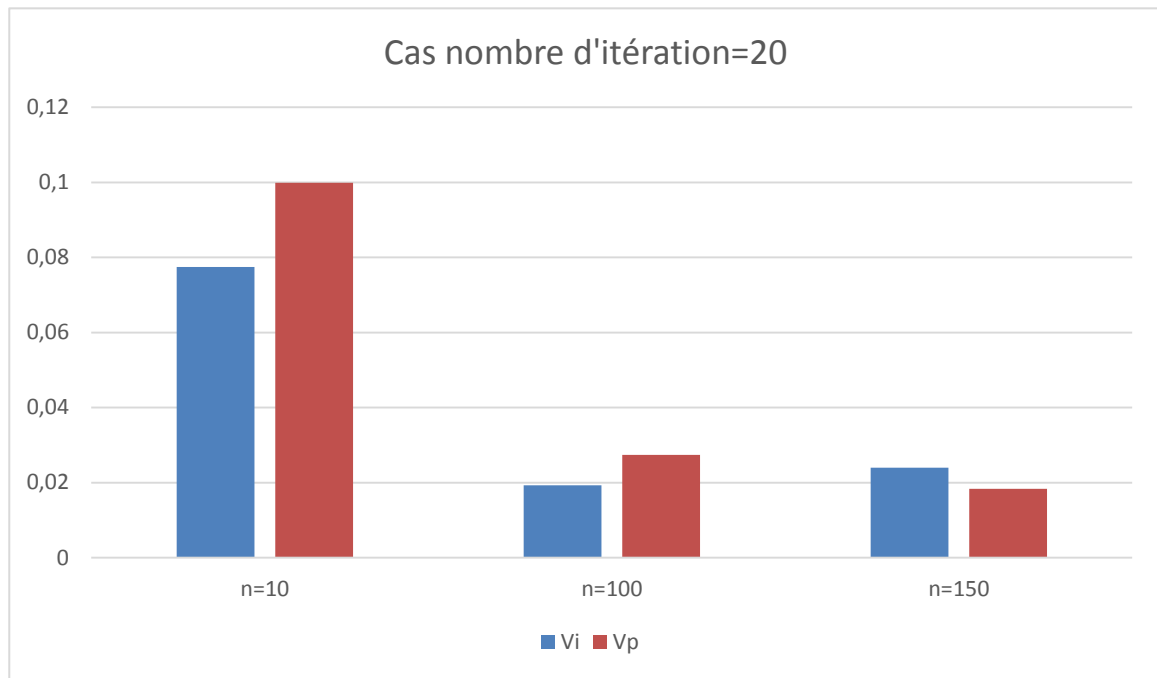


Figure (3.2) : Histogramme du rapport des moyennes pour comparer entre V_I et V_P .

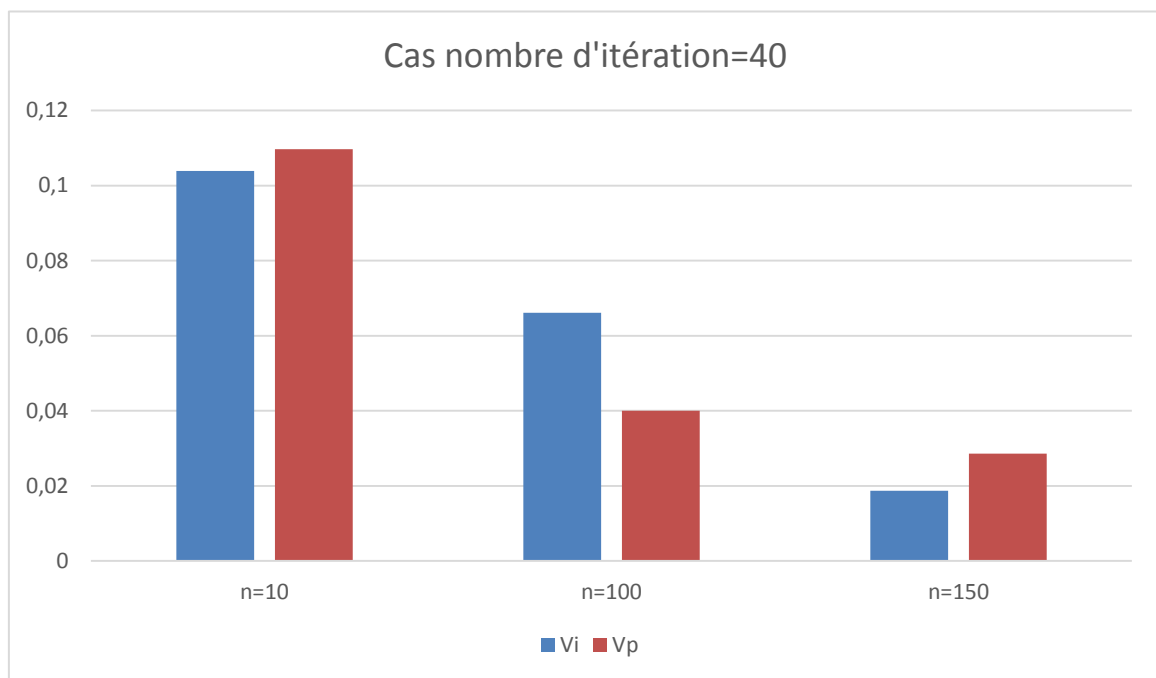


Figure (3.3) : Histogramme du rapport des moyennes pour comparer entre V_I et V_P .

Choix de la solution : aléatoirement (RANDOM).

$N_{it\acute{e}r} = 20$: Pour 20 itérations (Nombre d'itérations maximales).

$N_{it\acute{e}r} = 40$: Pour 40 itérations (Nombre d'itération maximales).

F_0 : La fonction objectif initial (RANDOM).

F_f : La fonction objectif final (optimal).

Moy : la moyenne.

Ra F : Le rapport d'amélioration parmi F_0 et F_f .

3.8. Commentaires et discussion des résultats

Grâce à notre étude des tableaux, nous notons :

Les résultats obtenus sont raisonnables. On remarque naturellement toujours l'amélioration nette de la valeur de la fonction objectif F de comparée à celle trouvée aléatoirement F_0 et ceci quel que soit le voisinage utilisées, Si l'augmentation de l'itération nous obtenons la solution optimale. Et puis avoir le temps à $N_{it\acute{e}r} = 40$ est supérieur au temps à $N_{it\acute{e}r} = 20$.

En généralement selon le rapport. On remarque que le voisinage par insertion (V_I) c'est mieux que du voisinage par permutation simple de deux tâches (V_P).

Nous notons la valeur de Ra F assez petit et cela montre une bonne amélioration de solution.

Du point de vue temps d'exécution V_P est meilleur surtout pour $n=10$ et $n=100$ dans le cas $N_{it\acute{e}r} = 20$.

3.9. Conclusion

Dans ce chapitre nous avons présenté une application de la recherche sur le problème d'ordonnement.

Nous avons choisi la stratégie de d'exploration (intensification) dans la méthode tabou destinée à restreindre la trajectoire de recherche à une partie de l'espace des solutions.

Les résultats obtenus sont satisfaisants. Ceci est en raison de la proportion des différences entre la première solution et la solution finale, après avoir analysé les résultats.

Nous pouvons conclure que le voisinage par insertion est mieux que du voisinage par permutation simple de deux tâches.