LA CONSTRUCTION DES HORAIRES JOURNALIERS MODELES PLNE

Résumé :

Dans la construction de plannings des salariés, si créer un planning optimisé d'une journée est aisé, mais créer un planning pour un mois est beaucoup plus complexe. En plus de la complexité combinatoire, il faut tenir compte de contraintes applicables sur les échelles hebdomadaire et mensuelle.

Afin de découpler la complexité combinatoire, la construction automatisée des plannings émet l'hypothèse que les plannings sont indépendants un jour sur l'autre. Ainsi, le problème se découpe en jours, enchaînes par la suite pour obtenir des plannings hebdomadaires. Les plannings sont élaborés en deux phases : la construction des vacations (ou horaires journaliers) puis l'enchaînement de ces journées dans la construction des tours mensuels. Voir par exemple [Bak76] ou [BM76]

Pour caractériser l'état de l'art en construction des plannings, ce chapitre évoquera d'abord les différents modèles pour chacune des deux phases. Ensuite il étudiera les méthodes de résolution de ces modèles par la Programmation Par Contraintes (PPC) ou par la Programmation Linéaire en Nombres Entiers, PLNE.

Les méthodes des recherches locales (y compris le recuit simulé, la recherche tabou, les algorithmes génétiques) et des algorithmes d'approximation seront également présentées. L'état de l'art sera complété par un rappel des systèmes interactifs d'aide à la décision.

Intuitivement, le problème est illustré par la Figure 2.1

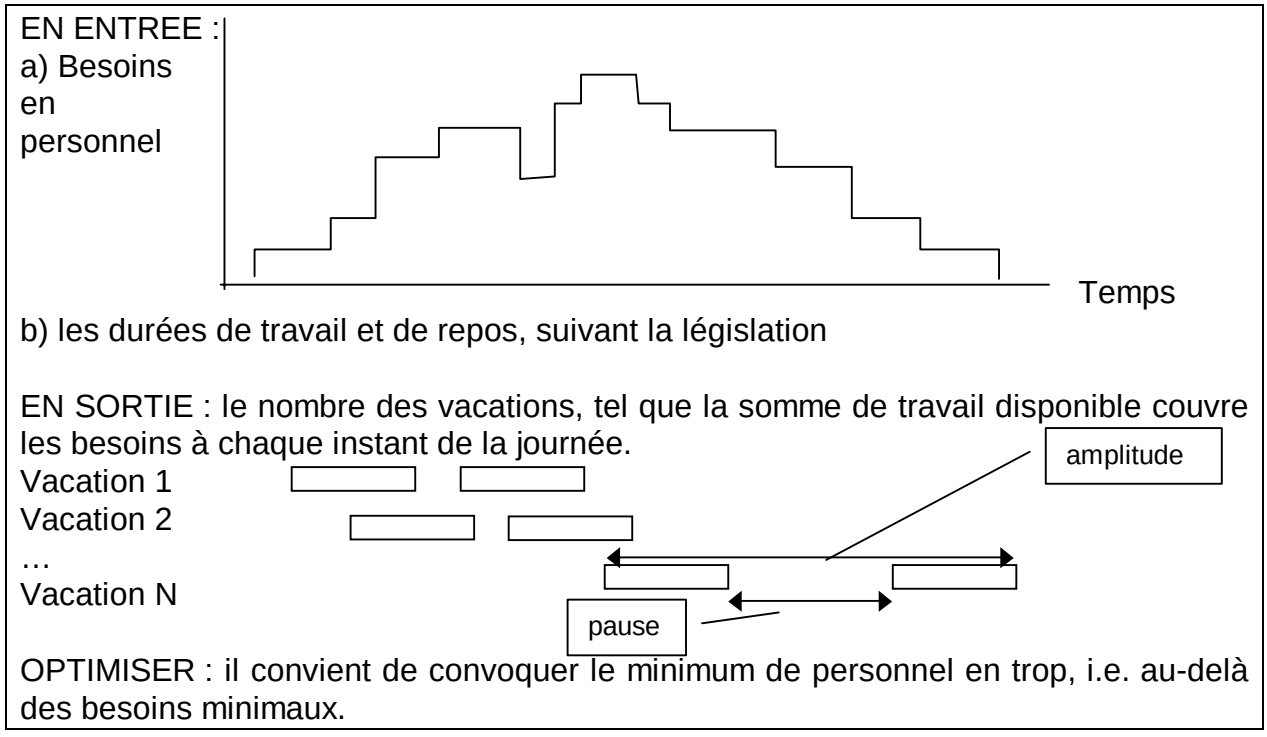


Figure 2.5. Construction des horaires journaliers

En entrée, on dispose des besoins en personnel : pour chaque intervalle de temps, la courbe de charge indique le nombre de personnes nécessaires. Cette information peut résulter des calculs statistiques sur le nombre d'appels par heure et le temps moyen de traitement par appel sur une période analogue.

En support du processus, on dispose d'un ensemble de règles légales qui définissent la journée de travail, ainsi qu'un ensemble de règles d'usage.

En sortie, on veut savoir comment faire travailler les salariés, soit la définition des horaires de début et fin et des horaires de pause, de telle sorte que la somme de leurs disponibilités couvre au mieux la courbe de charges avec un minimum de surplus.

2.1.1 Le modèle explicite de Dantzig

Le premier modèle de couverture de charge est dû à Dantzig dès 1954 [Dan54]. La construction des vacations se repose sur des modèles de couverture ensemblistes[3], décrit par les équations (1) et (2).

$$\text{Min } z = \sum_{j=1}^{m} C_j X_j \qquad (1)$$

$$\sum_{j=1}^{m} A_{ij} X_j \geq B_i \text{ , I=1,..,n et Xj} \geq 0\text{, entier} \qquad (2)$$

Où

n=nombre d'intervalles à couvrir dans l'horizon de planification,
m= nombre de vacations valides
Cj = coût lorsqu'un salarié est affecté à la vacation j
A_{ij} = 1 si l'intervalle i est couvert par la vacation j, 0 sinon
B_i = nombre de salaries requis pour l'intervalle i
X_j = nombre de salaries affectés à la vacation j

En amont du traitement par le modèle, l'ensemble des vacations valables est énuméré explicitement par rapport à l'ensemble des règles légales et d'usage. Exemple : la durée totale de travail ne doit pas dépasser 10 heures ; une pause ne débute qu'après 2 heures de travail effectif, etc.

Le modèle compte le nombre des salariés à chaque vacation, de telle sorte que les besoins par intervalle B sont satisfaits, tout en minimisant le coût total des affectations.

Exemple : Avec un seul type de vacation, sur une journée constituée de 9 intervalles d'une heure, les durées de la vacation sont de 4 à 7H

L'exemple comprend des vacations de durées de 4H (vacations a(j)), 5H : b(j), 6H : c(j) ou 7H : d(j), soit un total de 30 vacations.

[3] Generalized Set-Covering Problem GSCP

Temps:	1	2	3	4	5	6	7	8	9
a(1)	1	1	0	1	1				
a(2)		1	1	0	1	1			
a(3)			1	1	0	1	1		
a(4)				1	1	0	1	1	
a(5)					1	1	0	1	1
b(1)	1	1	0	1	1	1			
b(2)	1	1	1	0	1	1			
b(3)		1	1	0	1	1	1		
b(4)		1	1	1	0	1	1		
b(5)			1	1	0	1	1	1	
b(6)			1	1	1	0	1	1	
b(7)				1	1	0	1	1	1
b(8)				1	1	1	0	1	1
---									
c(1)	1	1	0	1	1	1	1		
c(2)	1	1	1	0	1	1	1		
c(3)	1	1	1	1	0	1	1		
c(4)		1	1	0	1	1	1	1	
c(5)		1	1	1	0	1	1	1	
c(6)		1	1	1	1	0	1	1	
c(7)			1	1	0	1	1	1	1
c(8)			1	1	1	0	1	1	1
c(9)			1	1	1	1	0	1	1
---									
d(1)	1	1	0	1	1	1	1	1	
d(2)	1	1	1	0	1	1	1	1	
d(3)	1	1	1	1	0	1	1	1	
d(4)	1	1	1	1	1	0	1	1	
d(5)		1	1	0	1	1	1	1	1
d(6)		1	1	1	0	1	1	1	1
d(7)		1	1	1	1	0	1	1	1
d(8)		1	1	1	1	1	0	1	1

Figure 2.6. Vacations avec pauses : 30 variables

Quant aux vacations sans pause sur une durée de 9H, il y a 6 vacations de 4H, 5 de 5H, 4 de 6H, 3 de 7H. Au total, on décompte 6+5+4+3+30 = 48 vacations.

Dans la pratique, un très grand nombre de vacations doit être utilisé afin de tenir compte de

- la souplesse inhérente à une charge définie au quart d'heure près
- des journées de longueurs différentes
- différents horaires de début, ou des pauses repas différentes

[Kla73] rapporte un nombre avoisinant 15000 vacations sur une journée, [BJ00] mentionne des milliards de horaires possibles sur une semaine. La génération de toutes les vacations pour la résolution avec ce modèle prendrait un certain temps. Chaque vacation étant représentée par une variable, les systèmes d'équations résultants sont trop grands pour être résolus par les logiciels disponibles aujourd'hui. D'où la motivation de la modélisation implicite des vacations.

2.1.2 Le modèle implicite de Moondra

Moondra fut le premier [Moo76] à proposer un modèle implicite où chaque vacation n'est plus représentée qu'implicitement par le nombre d'agents qui démarrent ou qui terminent au cours d'une intervalle p. Afin de traiter les différences des coûts, on crée des groupes pour les vacations ayant le même coût par période travaillée, la même pause repas et les mêmes restrictions en durée de travail (bornes sur la durée totale de la vacation et la durée travaillée sans interruption).

Variables de décision :
F(t, p)=nombre d'agents sur une vacation du type t qui se termine à la fin de l'intervalle p
S(t, p)=nombre d'agents sur une vacation du type t qui démarre au début de l'intervalle p

Comme le montre la figure suivante, le nombre d'agents présents à chaque intervalle p est donné par la somme des vacations ayant commencé avant ou pendant cette intervalle S(t, p), moins la somme des vacations déjà achevées. Pour notre exemple, le nombre de variables n'est que 9 par type de vacations de même coût.

Exemple : pour un seul type de vacations, sur 9 intervalles d'une heure. Avec des durées d'une vacation 5 à 7H, on ne commence plus au delà de i=5 donc S(t, p)=0 pour p > 5.

Nombre de personnes disponibles pendant l'intervalle i :
I = 1 : S(1,1)
I = 2 : S(1,1)+S(1,2)
I = 3 : S(1,1)+S(1,2)+S(1,3)
I = 4 : S(1,1)+S(1,2)+S(1,3)+S(1,4)
I = 5 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5)
I = 6 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) -F(1,5)
I = 7 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) -F(1,5)-F(1,6)
I = 8 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) -F(1,5)-F(1,6)-F(1,7)
I = 9 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) -F(1,5)-F(1,6)-F(1,7)-F(1,8)

Nombre global de personnes qui débutent et qui finissent
$S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5)=F(1,5)+F(1,6)+F(1,7)+F(1,8)+F(1,9)$

Pour limiter le minimum durée de travail à 5 périodes pour ceux qui commencent
en P=1 peuvent finir en P=5 : $S(1,1) \geq F(1,5)$
en 1 et 2 peuvent finir en 5 et 6 : $S(1,1)+S(1,2) \geq F(1,5)+F(1,6)$
en 1,2 et 3 peuvent finir en 5,6 et 7 : $S(1,1)+S(1,2)+S(1,3) \geq F(1,5)+F(1,6)+F(1,7)$
en 1,2,3 et 4 peuvent finir en 5,6,7 et 8:
$S(1,1)+S(1,2)+S(1,3)+S(1,4) \geq F(1,5)+F(1,6)+F(1,7)+F(1,8)$

Pour limiter le maximum durée de travail à 7 périodes pour ceux qui commencent
en 1 doivent finir en 5,6 et 7 : $S(1,1) \leq F(1,5)+F(1,6)+F(1,7)$
en 1 et 2 doivent finir en 5,6,7 et 8: $S(1,1)+S(1,2) \leq F(1,5)+F(1,6)+F(1,7)+F(1,8)$

Figure 2.7. Modèle Moondra avec 9 variables

Ce modèle est intéressant historiquement car c'est le premier modèle implicite. Avec les variables F et S, Moondra a proposé un système d'équations permettant de calculer le nombre d'agents présents pendant chaque période, qui doit couvrir les besoins, ainsi que des équations pour limiter la durée maximale et minimale des vacations. Ce modèle ne tient compte que d'une pause de durée fixe et d'une heure de début fixe par vacation.

2.1.3 Le modèle implicite de Bechtold et Jacobs

Le problème des pauses qui peuvent démarrer à des intervalles différents est résolu par [BJ90]. L'équivalence de ce modèle avec celui de Dantzig a été démontrée sous certaines conditions dans [BJ96]. Chaque type de vacation t est associé à une pause pendant une période de travail. Ce modèle utilise des variables B(i) qui donne le nombre de salariés

sur toutes les vacations dont la pause commence à une période i. Cette approche est limitée à une seule pause repas, dont la durée est la même pour toutes les vacations.

Variables de décision :
X(t) = nombre d'agents dans la vacation de type t
B(p) = nombre total de pauses initiées au début de la période p, quelle que soit la vacation type

L'apport de ce modèle est l'expression des contraintes qui garantissent que les personnes prendront leurs pauses comme prévu. Ces contraintes s'appuient sur l'hypothèse que les pauses ne sont pas imbriquées de façon *extraordinaire*, c'est à dire la fenêtre de pause d'un type de vacation j n'est pas strictement incluse dans celle d'un autre type j1.

2.1.4 Le modèle implicite de Thompson

Un modèle *doublement* implicite est proposé par [Tho95], permettant de tenir compte à la fois des heures de début de vacations de [Moo76] et les pauses repas de [BJ90]. Le nombre d'agents présents à chaque intervalle p est donné par la somme des vacations ayant commencé avant ou pendant cet intervalle S(t, p), moins la somme des vacations déjà achevées et des pauses en cours pendant cette période.

Variables de décision :
F(t, p)=nombre d'agents sur une vacation du type t qui se termine à la fin de la période p
S(t, p)=nombre d'agents sur une vacation du type t qui démarre au début de la période p
B(t, p)=nombre d'agents sur une vacation du type t, en pause commençant au début de p

L'avantage du modèle de Thompson est la souplesse qu'il offre dans l'étendue des vacations concernées, et surtout par la possibilité de trouver des solutions optimales.

Exemple : pour le même cas présenté ci-dessus
Nombre de personnes disponibles pendant l'intervalle i :

I = 1 : S(1,1)
I = 2 : S(1,1)+S(1,2)
I = 3 : S(1,1)+S(1,2)+S(1,3) -B(1,3)
I = 4 : S(1,1)+S(1,2)+S(1,3)+S(1,4) -B(1,4)
I = 5 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) -B(1,5)
I = 6 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) -F(1,5) -B(1,6)
I = 7 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) -F(1,5)-F(1,6) -B(1,7)
I = 8 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) -F(1,5)-F(1,6)-F(1,7)
I = 9 : S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) -F(1,5)-F(1,6)-F(1,7)-F(1,8)

Le nombre total de personnes qui travaillent égale le nombre total de pauses
S(1,1)+S(1,2)+S(1,3)+S(1,4)+S(1,5) = B(1,3)+B(1,4)+B(1,5)+B(1,6)+B(1,7)

La durée minimale d'une période de travail avant pause est de 2 intervalles :
Le nombre de personnes qui commencent en P=1 est au moins égal au nombre de pauses en P=3 ;
Sinon les pauses en P=3 comptent pour ceux qui commencent en P=2 $S(1,1) \geq B(1,3)$
Ceux qui commencent en P=1 et 2, doivent compter parmi ceux qui sont en pause en P=3 et 4 :

$S(1,1)+S(1,2) \geq B(1,3)+B(1,4)$
Idem : $S(1,1)+S(1,2)+S(1,3) \geq B(1,3)+B(1,4)+B(1,5)$
$S(1,1)+S(1,2)+S(1,3)+S(1,4) \geq B(1,3)+B(1,4)+B(1,5)+B(1,6)$

Pour limiter la durée d'une période de travail après pause à 2 intervalles :

$F(1,9) \geq B(1,7)$
$F(1,8)+F(1,9) \geq B(1,6)+B(1,7)$
$F(1,7)+F(1,8)+F(1,9) \geq B(1,5)+B(1,6)+B(1,7)$
$F(1,6)+F(1,7)+F(1,8)+F(1,9) \geq B(1,4)+B(1,5)+B(1,6)+B(1,7)$

Les restrictions sur la durée maximale d'une période de travail à 4 intervalles ne sont pas nécessaires dans ce cas. Avec une durée maximale de travail à 7 intervalles, une pause repas à 1 intervalle, la durée minimum à 2 intervalles, on obtient une durée maximale à 4.

Figure 2.8. Modèle de Thompson : 14 variables

Un autre modèle implicite proposé par [Ayk96], traite la situation où existent des pauses supplémentaires avant et après la pause repas. Les variables sont les suivantes :

Variables de décision :
U(t, p)= nombre de salariés affectés à une vacation type t dont la première pause courte aura lieu à la période p
W(t, p)= nombre de salariés affectés à une vacation type t dont la pause repas = p
B(t, p)= nombre de salariés affectés à une vacation type t dont la deuxième pause courte = p

Compte tenu des variables U, W et V, le nombre de personnes disponible au cours de la période p, X'(t, p) est donné par :
X'(t, p) = X(p) si p n'est pas une pause
X'(t, p) = X(p) – U(t, p) si p est la première pause courte
X'(t, p) = X(p) – W(t, p) si p est la pause repas et p-1 n'est pas la pause repas
X'(t, p) = X(p) – W(t, p) – W(t,p-1) si p et p-1 sont les pauses repas
X'(t, p) = X(p) – W(t, p-1) si p-1 est la pause repas et p n'est pas la pause repas
X'(t, p) = X(p) – V(t, p) si p est la deuxième pause courte

Contrainte de couverture de charge (proche du Modèle de Danzig) :
A(t, p) = 1 si la période p est travaillée dans la vacation de type t, = 0 sinon
$\sum_{t=1,n} A(t, p) * X'(t, p) \geq R(p)$ pour chaque période p

Figure 2.9. Modèle d'Aykins [Ayk96]

2.1.5 Le modèle implicite de Jarrah

Le modèle de Jarrah [JBS94] a intégré la modélisation implicite des pauses de [BJ90], avec les algorithmes exacts de positionnement de repos développés dans [BC85], et de plus traite les travailleurs à temps partiel.

Variables de décision :
X(t) = nombre d'agents temps plein dans la vacation de type t
Y(t) = nombre d'agents temps partiel dans la vacation de type t
B(i) = nombre total de pauses initiées au début de la période i, quelle que soit la vacation type

F(j, t) = 1 si la vacation en temps plein type j couvre la période t, 0 sinon
P(j, t) = 1 si la vacation en temps partiel type j couvre la période t, 0 sinon

Figure 2.10. Modèle de Jarrah et al.

Utilise les variables:
p = ratio des salariés en temps plein aux salariés en temps partiel.
Nf = nombre de types de vacations à temps plein
Np = nombre de types de vacations à temps partial
R(t)= besoins en personnel au cours de la période t

F(j, t) et P(j, T) sont utilisés dans la formulation de couverture ensembliste généralisée :

$$\sum_{j=1, Nf} F(j, t) * X(t) + \sum_{j=1, Np} P(j, t) * Y(t) - B(t) \geq R(t) \quad (3)$$

Suite à la résolution du PLNE, il faut affecter les personnes suivant leur contrat temps. Un algorithme glouton est proposé par Jarrah.

2.1.6 Modèle implicite de Cai et Li

A titre de comparaison, nous citons un modèle traitant le personnel à multiples compétences [CL00]. On considère trois populations d'employées qualifiées en 1, qualifiées en 2 et qualifiées en 1 et 2.

Variables de décision :
X(j) = nombre d'agents de qualification 1 dans la vacation j
Y(j) = nombre d'agents de qualification 2 dans la vacation j
Z1(j) = no. d'agents de qualification 1+2 dans la vacation j faisant le travail qualification 1
Z2(j) = no. d'agents de qualification 1+2 dans la vacation j faisant le travail qualification 2

Contrainte de couverture de charge (Modèle de Danzig) :
A(j, p) = 1 si la période p est travaillée dans la vacation de type t, = 0 sinon
$\sum_{j=1,n} A(j, p) * X(j) + \sum_{j=1,n} A(j, p) * Z1(j) \geq R1(p)$ pour chaque période p, besoins 1
$\sum_{j=1,n} A(j, p) * Y(j) + \sum_{j=1,n} A(j, p) * Z2(j) \geq R2(p)$ pour chaque période p, besoins 2

Figure 2.11. Modèle de Cai et Li [CL00]

L'intérêt de ce modèle est la prise en compte des multiples compétences. Le modèle résultant exige un grand nombre de variables, car il faut considérer toutes les combinaisons de compétences.

Pour résoudre ce modèle, les auteurs proposent une méthode à base de l'algorithme génétique. La durée des intervalles est l'heure ; pour les vacations jour, on peut commencer entre 7 H et 15H et il n'y a qu'une vacation nuit (au total 10 vacations).

2.1.7 Conclusion sur les modèles PLNE

Les modèles en Programmation Linéaire peuvent exprimer les contraintes sur les durées de travail journalier. Les variables en nombres entiers donnent le nombre de salariés qui effectuent un travail caractérisé, ainsi la méthode peut traiter un grand nombre de salariés **homogènes**.

Les travaux ont montré que les modèles implicites peuvent être résolus très efficacement avec la PLNE : de façon générale, les instances donnent des solutions entières optimales.

2.2 CONSTRUCTION DES HORAIRES JOURNALIERS : MODELES PPC

Ce paragraphe présente des modèles permettant de résoudre le problème du paragraphe précédent avec la Programmation Par Contraintes.

2.2.1 Le modèle explicite de Partouche

Dans ce modèle, la liste de vacations candidates est représentée explicitement comme un objet contenant 3 vecteurs de même longueur : le vecteur des heures de début, le vecteur des heures de fin et le vecteur des durées.

Variables de décision :
X(j) = nombre d'agents pris dans la vacation j, une variable dont le domaine est {1..n}, ce qui assure qu'elle aura une valeur entière
A(j, p) = X(j) si la période p est couverte par la vacation j, = 0 sinon

Contrainte de couverture de charge :
Pour chaque période p, $\sum_{j=1,n} A(j, p)$ doit couvrir les besoins B(p)
Minimiser la somme des excédents e(p) = $\sum_{j=1,n} A(j, p) - B(p)$
Contraintes supplémentaires :
Non-symétrie parmi les agents
Le nombre d'agents est réduit au strict minimum i.e. $\text{Max}_p B(p)$

Figure 2.12. Modèle explicite de Partouche

D'après [Par98], ce modèle est très peu performant en PPC même pour des tests comportant seulement de 10 à 48 agents.

2.2.2 Le modèle implicite de Partouche-Moondra

[Par98] présenta un deuxième modèle implicite de planning individuel avec des variables de début et de fin de vacation, proche de celui de Moondra, ce qui évite d'énumérer toutes les vacations possibles. Pour chaque agent, le système construit sa vacation définie par l'heure de début et de fin.

Variables de décision :
Y(j) = 1 si la vacation j est choisie dans la solution , = 0 sinon
S(j) = heure de début de la vacation j,
F(j) = heure de fin de la vacation j

Variables dépendantes :
La matrice A(i, j) qui vaudra 1 si l'intervalle i est couvert par la vacation j
A(i, j) = 1 si (Y(j) = 1) et ((S(j) ≤ i) et (F(j) > i))
Les amplitudes des vacations D(j) = S(j) – F(j) dont le domaine est prédéfini.

Contrainte de couverture de charge :
La somme pour chaque intervalle i, $\sum_{j=1,n} A(i, j)$ doit couvrir les besoins B(i) et on cherche à minimiser la somme des excédents $e(i) = \sum_{j=1,n} A(i, j) – B(i)$

Contrainte supplémentaire : Asymétrie
Construire les vacations dans l'ordre chronologique : Pour tout j > 1, S(j) ≥ S(j-1)

Stratégie
Les variables Y, F et S sont affectées dans cet ordre.

Figure 2.13. Modèle implicite de Partouche – Moondra

Avec ce modèle, la première solution est souvent trouvée très rapidement, les résultats sont globalement décevants au niveau de l'optimum. Les résultats suivants ont été publiés pour un PC 486 à 100 M Hz.

	PLNE		PPC 1er		PPC 2ème	
	CPU	Objectif	CPU	Objectif	CPU	Objectif
Test1 10 pers.	<1 s.	2	41 s.	2	4 s.	2
Test 2.1 48 per	<1 s.	171	> 20 min.	Pas de soln.	7 s.	171*+
Test 2.2	<1 s.	211	1 s.	1377*	7 s.	225*
Test 2.3	<1 s.	447		560*	6 s.	474*

Légende : * première solution, + optimum non prouvé

En effet, Partouche propose des plannings individuels (avec 3 variables par individu). Ce type de modèle permet de prendre en compte les particularités individuelles (par ex. préférences et historique). Les modèles planning anonyme utilisent des variables qui donnent le nombre de salariés d'un tel type : ils peuvent traiter des situations avec un nombre plus grand de salariés mais ils ne peuvent pas prendre en compte les préférences et historique.

2.2.3 Notre modèle utilisant la contrainte cumulative

Avec des contraintes locales, les modèles précédents sont peu efficaces. Pour obtenir un maximum de propagation, il est vivement conseillé d'utiliser les contraintes globales. Nous présenterons la contrainte cumulative pour la construction des horaires journaliers. La contrainte cumulative [AB93] a été conçue pour résoudre le problème de l'ordonnancement des tâches où le nombre de ressources disponibles est limité. La définition de la contrainte est donné dans le glossaire.

Il n'est pas évident de savoir comment elle peut servir car l'opérateur de comparaison dans la couverture des charges est supérieure ou égal et non inférieure ou égale.

[SC95] montrait comment utiliser la contrainte cumulative pour modéliser le concept du producteur/consommateur, dans lequel le ou les processus produisent une commodité en stock et des processus qui le consomment, chaque processus ayant lieu à des instants et avec des quantités précises. Voir la Figure 2.14.

Soit

N le nombre d'événements producteurs P_1, …, P_n,
qui produisent la quantité Q_{Pi} au moment T_{Pi}
M le nombre d'événements consommateurs C_1, …, C_m
qui consomment Q_{Cj} au moment T_{Cj}
Q_0 le niveau initial du stock
$a = \min \{ T_{Cj}, T_{Pi} \} - 1$ et $b = \max \{ T_{Cj}, T_{Pi} \} + 1$

Contrainte de producteur/consommateur :

$$\forall t \in [a, b] \quad Q_0 + \sum_{T_{Pi} \le t} Q_{Pi} \ge \sum_{T_{Cj} \le t} Q_{Cj} \qquad (4)$$

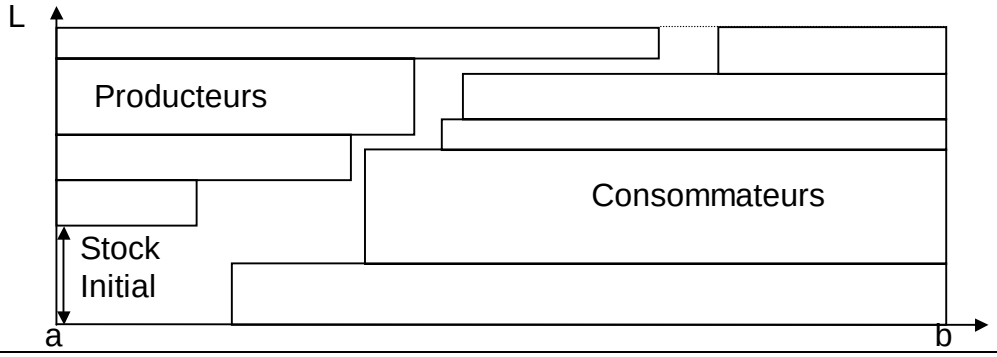
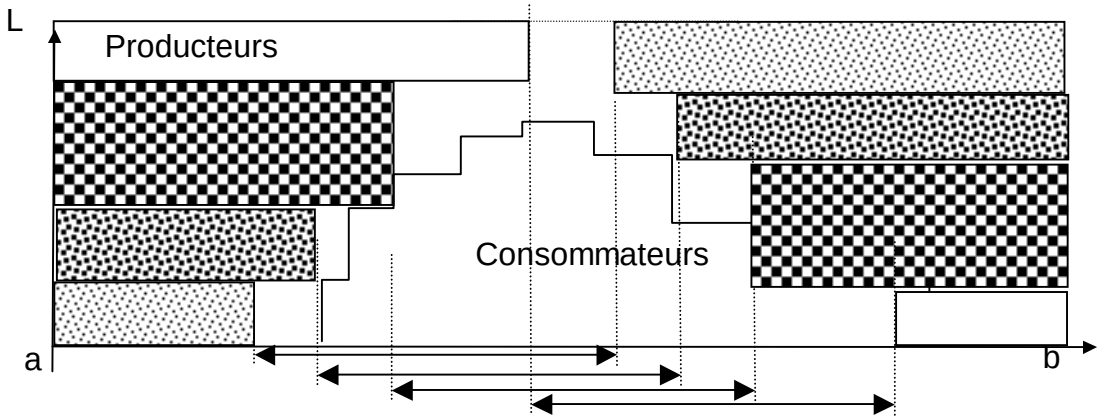


Figure 2.14. Définition de la contrainte Producteur/Consommateur

Le raisonnement suivant a été proposé pour formuler ce problème à l'aide de la contrainte cumulative. Un processus producteur P_i rend disponible la quantité Q_{Pi} de ressource au moment T_{Pi} : on l'exprime avec une tâche cumulative qui occupe la ressource depuis le début jusqu'à T_{Pi}. Inversement, un processus C_j consomme la quantité Q_{Cj} de ressource au moment T_{Cj} : on l'exprime avec une tâche cumulative qui occupe la ressource depuis T_{Cj} jusqu'à la fin de l'horizon de calcul.

Nous nous sommes s'inspiré de ce travail pour créer un modèle de couverture des charges avec la contrainte cumulative où $Q_{Pi} = 1$ et $Q_{Cj} = 1$. Un employé i qui travaille de T_{Di} à T_{Fi}, rend disponible une ressource pendant la durée de sa présence. On l'exprime avec deux tâches : l'une dure depuis le début jusqu'à T_{Di} et l'autre depuis T_{Fi} jusqu'à la fin. La courbe de charges peut simplement se définir par un ensemble de tâches



Début et fin des vacations de chaque groupe d'employés

Figure 2.15. Adaptation à la couverture de la courbe des charges

2.2.4 Autres modèles implicites

Nous proposons de résumer ici un modèle implicite de construction des vacations, dans le contexte des multiples qualifications. Il sera utilisé dans des développements décrits au chapitre 5.

Variables de décision :
X(e, i) = q où e est un employé, i est un intervalle de temps, et q la qualification de l'employé pendant cet intervalle, ou l'affectation au repos.

Ainsi l'ensemble des vacations n'est pas énuméré avant la résolution des charges de couverture.

Contrainte de couverture de charges :
$\forall i \in 1, \ldots I, \forall q \in$ **Qualifications**, $W_D(i, q) \leq |\{ X(e, i) = q, \forall e \in$ **Employés** $\}|$
Utilisation des primitifs PPC décrits au glossaire : voir [BSK+97b]
among($[N_1, ..., N_Q]$, $[X(e_1, i), \ldots, X(e_N, i)]$, Zeros $_N$, $[q_1, ..., q_Q]$, all)
Minimiser la somme des excédents = $\sum_q \sum_i |\{ X(e, i) = q, \forall e \}| - W(i, q)$

Contrainte sur la durée totale de travail pour un employé e :
Durée totale du travail : among ([Min, Max], $[X(e, 1), \ldots, X(e, I)]$, Zeros $_I$, [repos], all)
on restreint le nombre total de variables prenant la valeur repos

Contrainte sur les pauses repas (après 6H consécutives de travail)
Les contraintes de repos journalier sont implantées dès que la fin de la journée précédente est connue, on peut déterminer le début au plus tôt le lendemain.

Figure 2.16. Autre modèle implicite

Au chapitre 5, nous proposerons une méthode de résolution utilisant ce modèle à multiples qualifications.

2.2.5 Conclusion sur la construction de vacations par la PPC

Les modèles conçus pour la PLNE ne peuvent pas être traités avec la même efficacité en PPC, car les contraintes de type *somme* présentes dans ces modèles sont de nature globale et non locale à un nombre de variables. La propagation locale est très faible, ce qui conduit à la conclusion de Partouche que la PPC n'est pas adaptée pour résoudre ce problème (cf. [Par98]). Nous avons proposé un modèle utilisant la contrainte cumulative permettant de tirer un maximum de propagation.

2.3 LA CONSTRUCTION DE TOURS ACYCLIQUES : MODELES PLNE

Suite à la constitution d'une courbe de charge journalière, le calcul de dimensionnement de l'équipe et la construction des vacations, nous connaissons le besoin en nombre de personnes à chaque type de vacation pour chaque jour de la semaine. La création de plannings mensuels (ou construction de tours) peut se faire de façon cyclique ou non.

Nous proposons de faire le tour des méthodes usuelles de planification acyclique.

2.3.1 Calcul des Journées Repos. Day-Off Scheduling

Lorsque la durée de travail hebdomadaire d'un salarié ne correspond pas à la semaine œuvrée de l'entreprise, il faut mettre en place le calcul des journées de repos. Ex. la semaine de 5 jours dans des entreprises ouvertes 7 jours par semaine dans le cas d'un salarié à temps plein.

Différents cas de repos hebdomadaires peuvent être envisagés :

- 2 jours de repos par semaine
- 2 jours consécutifs par semaine
- 4 jours de repos sur 2 semaines, dont 2 jours consécutifs
- 4 jours de repos sur 2 semaines, dont 1 week-end

Exemple d'un modèle PLNE pour le calcul des journées repos

Soit les besoins B(p) pour les périodes p=1, …, 7, et l'hypothèse de deux jours de repos consécutifs par semaine pour chaque employé, le programme linéaire correspondant est :

Minimiser $\sum X(p)$

$$\begin{pmatrix} 0&1&1&1&1&1&0\\ 0&0&1&1&1&1&1\\ 1&0&0&1&1&1&1\\ 1&1&0&0&1&1&1\\ 1&1&1&0&0&1&1\\ 1&1&1&1&0&0&1\\ 1&1&1&1&1&0&0 \end{pmatrix} \begin{pmatrix} X(1)\\ X(2)\\ X(3)\\ X(4)\\ X(5)\\ X(6)\\ X(7) \end{pmatrix} \geq \begin{pmatrix} B(1)\\ B(2)\\ B(3)\\ B(4)\\ B(5)\\ B(6)\\ B(7) \end{pmatrix}$$

Figure 2.17 Calcul des journées de repos

Des matrices semblables peuvent être proposées pour les autres cas de repos hebdomadaires. [Bak74] a analysé le cas particulier où

1. la somme des besoins est un multiple de 5
2. Le maximum des besoins est inférieur au cinquième de la somme des besoins

Sans la condition 1, il n'est pas possible qu'un effectif entier couvre **exactement** la charge. Sans la condition 2, l'effectif nécessaire pour couvrir la charge maximale engendre immanquablement une couverture totale supérieure à la charge totale.

Lorsque ces deux conditions sont vérifiées, il peut exister une solution avec l'effectif égal au cinquième de la somme des besoins, qui ne génère aucun sur-effectif.

[BR78] suggère le problème complémentaire du problème initial qui est en quelque sorte l'image inversée de l'original :

Maximiser $\sum Y(p)$

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} Y(1) \\ Y(2) \\ Y(3) \\ Y(4) \\ Y(5) \\ Y(6) \\ Y(7) \end{pmatrix} \leq \begin{pmatrix} W\text{-}B(1) \\ W\text{-}B(2) \\ W\text{-}B(3) \\ W\text{-}B(4) \\ W\text{-}B(5) \\ W\text{-}B(6) \\ W\text{-}B(7) \end{pmatrix}$$

Figure 2.18 Calcul des journées de repos : matrice inverse

Si W correspond à l'effectif total, W-B(p) correspond au nombre maximum de salariés pouvant être au repos à la période p. Le problème consiste donc à maximiser le nombre d'agents ayant deux repos consécutifs chaque semaine.

L'intérêt de ce modèle réside par la traduction via le programme linéaire en Y d'un problème de couplage maximal dans un graphe constitué d'un cycle simple de 7 nœuds.

Conclusion sur le calcul des journées repos

L'accord des journées repos est un fait éminemment contractuel ou syndical. Dans un milieu où il est difficile de trouver des candidats avec les compétences requises pour étoffer les équipes telles que les équipes de soins médicalisés, il faudra offrir des garanties sur le positionnement des repos.

Ce calcul se fait bien puisque le modèle est très petit.

2.3.2 La construction des tours

La construction des vacations fournit le nombre des vacations à fournir par jour afin de couvrir les besoins. Il n'affecte pas de vacations aux salariés. Il faut alors construire des tours et les affecter aux salariés.

Méthode de résolution utilisant la PLNE

Les variables de décision sont

$X(e, t, k) = 1$ si le jour t, l'employé est affecté à l'étiquette k, = 0 sinon

$X_R(e, t) = 1$ si le jour t est affecté au repos, = 0 sinon

Les contraintes légales sont :

- Contrainte d'unicité : $\sum_k X(e, t, k) \leq 1, \forall (t, e)$
  On place au plus une vacation à chaque jour de l'horizon pour l'employé
- Couverture des besoins : : $\sum_e X(e, t, k) \leq v_{tk}, \forall (t, k)$
- Au plus N vacations k, toutes les P semaines
  Sur toutes les variables t=7*w,…, 7*w + 7*P-1 pour la semaine w
  $\sum_{t = 7*w, \ldots, 7*w+7*P-1} X(e, t, k) \leq N, \forall e, w$ (**5**)

- Au plus C jours successifs de travail : sur C+1 jours consécutifs, on a au moins un repos. $\sum_{i=0, C} X_R (e, t + i) \geq 1, \forall t$ (**6**)
- Repos journalier: si k=0 représente le repos,
  $X(e, t, k) + X(e, t+1, k') \leq 1$ avec $k <> 0, k' < k \ (\forall e, k)$ (**7**)

Afin de réaliser avec la PLNE, une contrainte de transition du type :

si X(e, t, k) = 1 alors X(e, t+1, k') = 1,

[Wil91] propose d'utiliser une variable indicateur δ et une linéaire équation du type : $X - \delta \leq 0$ qui est équivalent à { $X=0 \equiv \delta=0$ & $X=1 \equiv \delta=1$ }

$X(e, t, k) - X(e, t+1, k') \leq 0$

Pour tenir compte des pré-affectations aux employés, il suffit d'ajouter des équations du type X(e, t, k) = 1 dans ce modèle. Si une personne ne peut pas être affectée au code k' : il suffit d'ajouter la contrainte X(e, t, k') = 0.

Méthode de résolution avec des modèles de flot

Afin de produire des tours hebdomadaires, [CGL01] utilise une formulation en PLNE pour combiner les modèles journaliers dans un cadre de flot. Le principe consiste à construire un réseau dans lequel un chemin du nœud source au nœud puit respecte les contraintes suivantes :

T1 : règle sur les jours de repos (ex. au moins un jour doit être un week-end)

T2 : la différence des heures débutantes des jours consécutifs d'un tour n'excède pas une limite donnée (ex. deux heures)

T3 : les tours peuvent être cycliques (ne pas violer la contrainte sur les heures débutantes pour le dernier et le premier jours).

Un chemin entre les nœuds s et f représente un tour qui satisfait les contraintes sur les jours de repos, en visitant 5 nœuds W (jours travaillés obtenu de la construction de vacations journalières) et 2 nœuds F (jours repos).

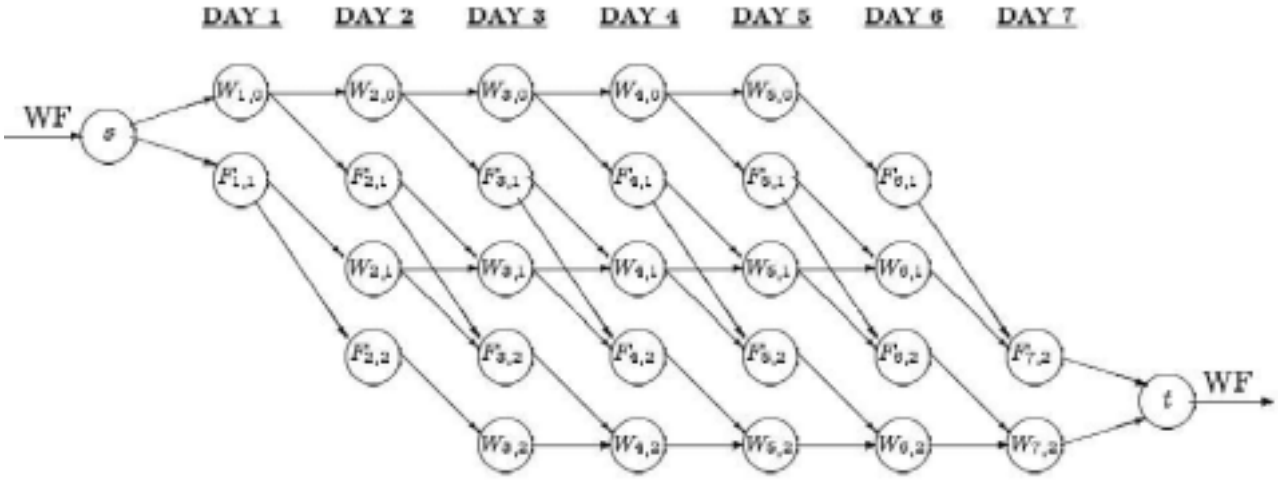


Figure 2.19 Modèle de flot hebdomadaire, repris de [CGL01]

Un arc est représenté entre deux nœuds si la transition correspondante est admise. Par exemple le flot du nœud $F_{3,1}$ to $W_{4,1}$ correspond au nombre des agents qui ont leur premier jour de repos le jour 3 et qui travaillent le jour 4.

2.3.3 La construction intégrée des vacations et tours

[BJ00] proposent un modèle implicite capable de traiter des intervalles au quart d'heure sur un horizon d'une semaine. Le nombre de variables dans le modèle explicite correspondant serait de l'ordre du milliard ! Il exploite l'idée de « time bands ». Au cours d'une même semaine, un salarié ne doit démarrer sa journée que dans une plage de temps. D'une part, cette condition génère des plannings très appréciés par tous et d'autre part, elle permet une réduction énorme de la combinatoire.

2.4 LA CONSTRUCTION DE TOURS : MODELES PPC

Nous présentons dans ce paragraphe quelques généralités sur la construction de tours avec des modèles PPC. Ce sujet sera traité en détails au chapitre 3.

2.4.1 Modèles Primal et Dual

La construction de tours utilise le modèle suivant : les variables de décision sont des affectations d'un employé un jour donné et les valeurs possibles sont les horaires. Voir par ex. [CGL95], [Heu96] ou [Par98].

X(t, e) = k si le jour t l'employé est affecté à l'étiquette k ou à l'étiquette repos.
Besoins(t, k) = Nombre d'étiquettes nécessaires le jour t

L'avantage de ce modèle est que la contrainte d'unicité est implicitement respectée : chaque salarié a une et une seule affectation par jour. Les outils PPC offrent les différentes contraintes suivantes :

- **all_different** ($[X_1, \ldots, X_N]$), qui exige que toutes les variables X_i aient des valeurs différentes deux à deux.
- **at_most**.(M, $[X_1, \ldots, X_N]$) qui exige que la valeur maximale des variables X_i soit égale ou inférieure à M.
- **cumulative, among** et **sequence** seront présentées dans le glossaire.

La contrainte de charge est modélisée avec la contrainte globale **among** pour le jour j :

among([Min, Max], [X(e_1, i), …, X(e_N, i)], Zeros $_N$, [k], all)

Cet appel spécifie que le nombre de variables dans la liste d'affectations pour le jour j pouvant prendre la valeur k doit être compris entre les bornes Min et Max.

En général, les outils PPC proposent des propagations locales de type '**si … alors**'. Par ailleurs [MGS96] propose une intégration des règles et des contraintes pour résoudre les problèmes d'affectation.

Le modèle dual a été proposé par [Tsa93] où les variables représentent des horaires et les valeurs représentent les salariés:

Y(t, k, i) $\in$ Employés, pour i = 1 , …, Besoins(t, k)
All_different({ Y(t, k, 1), …, Y(t, k, Besoins(t, k)) }), $\forall$ (t, k)

[MGS96] proposent une application utilisant ces variables, comme le montre le planning suivant :

t	k	i				
Etiquette	Poste	Jour 1	Jour 2	Jour 3	…	Jour N
Matin	Responsable	Qui ?				
Matin	Infirmière DE 1					
Matin	Infirmière DE 2					
Matin	Aide Soignante 1					
Matin	Aide Soignante 2					
Soir	Responsable					
Soir	Infirmière DE 1					
Soir	Aide Soignante 1					
Nuit	Responsable					
Nuit	Infirmière DE 1					

Figure 2.20 Un planning avec des variables dual

Initialement, le domaine des variables contient tous les salariés disponibles ce jour là et ayant les compétences requises. La taille du domaine des variables indique la difficulté d'affectation de cet horaire/poste/jour. Certaines variables peuvent avoir un domaine réduit à une seule valeur : il faut alors réaliser l'affectation afin de conserver la consistance du problème.

2.4.2 La construction simultanée des repos et tours

Pour résoudre en un seul processus le calcul des journées repos et l'enchaînement des vacations sur la semaine ou le mois, les applications Gymnaste et Equitime décrites au chapitre 3 effectuent ce calcul simultané. L'objectif est de proposer des affectations de journées (y compris le repos hebdomadaire) de façon équitable pour tous les salariés.

2.4.3 Une comparaison des modèles PLNE et PPC

Les paragraphes précédents ont présenté des modèles PLNE et PPC pour construire des vacations et des tours. Dans ce paragraphe, nous présentons des observations générales sur ces deux types de modèles.

Richesse sémantique

Les variables PPC sont du type X(e, t) = k, alors que les variables PLNE sont X(e, t, k) = 1 ou 0 (où l'employé e, t représente la date et la vacation k).

Les variables PPC sont plus proches des pensées du planificateur et de sa terminologie. La valeur des variables PPC, est plus riche sémantiquement que les valeurs booléennes des variables PLNE. Ainsi le modèle PPC a besoin de moins de variables, dans le rapport 1 pour |Vacations|. En plus, les contraintes « d'affectation unique » par salarié par jour sont implicitement respectées.

Avec des variables plus *riches*, la PPC permet de proposer des outils de contraintes très expressifs, qui dépassent les limites des inéquations linéaires de la PLNE. Les algorithmes de consistance ont une sémantique beaucoup plus proche du problème à résoudre que celle de l'algorithme du simplexe par exemple. « Cela tient sans doute aux origines Intelligence Artificielle de la PPC », comme l'écrivait Partouche [Par98].

Avec des variables plus *riches*, la PPC admet des variables dual en construction de tours : elles permettent des déductions supplémentaires, lorsque des contraintes portant sur ces variables sont applicables.

Anonyme ou individuel

Les modèles PLNE sont en général anonymes, et calculent le nombre de salariés dans des groupes homogènes qui sont pris en compte par les modèles. De ce fait, ils sont capables de traiter un grand nombre de salariés. Par ex. des salariés temps plein et à temps partiel, des salariés prenant une pause débutant à midi, etc.

Par contre, les modèles PPC sont en général individuels et ainsi peuvent prendre en compte les spécificités de chaque salarié. Suivant la situation locale de résolution, il est possible de programmer des déductions spécifiques, à la manière des règles.

> Ex : Si Agent 1 travaille le samedi, alors il sera de repos le lundi suivant.
> (uniquement pour cet agent)

Ce constat ne met pas en cause les techniques de résolution, mais découle des origines intellectuels de leurs concepteurs. L'origine IA des concepteurs pratiquant la PPC donne la priorité au traitement des individuels. L'origine de la RO étant liée à l'effort de la guerre, les concepteurs donnent la priorité aux masses.

PLNE : la recherche des solutions entières

Il est connu qu'en général, les solutions sont continues. Pour obtenir des solutions entières, il faut exécuter des algorithmes de recherche. Les matrices uni-modales admettent des solutions entières.

Optimisation

Les modèles PLNE donnent la possibilité d'optimiser une fonction de coût, à l'insu des modèles PPC. Généralement, les chercheurs utilisent des fonctions économiques en termes de salaires ou primes. Cependant, il est notoire que la formulation de cette fonction est très difficile car elle est complexe *politiquement*[4]. Il n'est pas évident de comparer le coût d'un travail de nuit pour une personne très expérimentée au travail d'un débutant le week-end. Cette fonction ne peut pas prendre en compte ni la productivité ni la motivation des individus, simplement parce que le modèle est anonyme.

Pour le problème de programmation des cours à l'université, [KW92] a écrit :

[4] Cela constitue des problèmes mal définis, décrit dans le Glossaire. Ill-defined problems.

"The objective function approach is generally unsuitable for problems of this complexity due to the difficulty or impossibility of quantifying some of the factors".

Faiblesse de la propagation locale

Caseau et al. écrivaient [CGL95] :

"Timetabling problems are hard to solve with constraint logic programming. After reporting impressive successes for job-shop scheduling, CLP users have tried to address timetabling problems with far less success. The key is that local propagation is the right tool for job-shop scheduling, whereas timetabling scheduling is dominated by the combinations of global constraints that cannot be solved efficiently by local propagation..".

2.5 LA CONSTRUCTION DE TOURS CYCLIQUES

Connaissant le nombre de vacations par jour, trouvées par la *construction des vacations*, on peut concevoir un cycle qui fournit les dites vacations tout en respectant les règles d'enchaînement des vacations et autres critères.

En déroulant le cycle, on obtient ainsi un planning prévisionnel *théorique*. Ce planning peut être modifié manuellement pour s'adapter aux congés annuels. Cela donne le planning prévisionnel *ajusté*. Dans certaines entreprises, il peut avoir plusieurs niveaux de plannings prévisionnel ajusté afin de tenir en compte des événements initialement non prévus, qui se déclarent au fur et à mesure.

2.5.1 Types de cycles

Les cycles peuvent être regroupés par leurs durées :

- Des cycles journaliers utilisés au ministère de la justice seront cités en exemple : une séquence de N vacations incluant des jours de repos hebdomadaire. Les salariés bénéficient de peu de week-ends sur un an. Ex. Cycle de 5 et de 6 jours :

Cycle 5 jours	Matin	Soir	Nuit	Repos	Repos	
Cycle 6 jours	Matin	Soir	Soir	Nuit	Repos	Repos

  Le repos après le travail de nuit est un repos obligatoire.

- Des cycles hebdomadaires (la période étant un nombre de semaines entières) sont très souvent proposés, car cela permet de caler les repos hebdomadaires sur les week-ends.

  Un Cycle Hebdomadaire de N semaines est défini par une grille de codes horaires sur N semaines. Prenons comme exemple un cycle de 5 semaines proposé par [LNB80], avec une de chaque vacation M=matin, S=Soir, N=Nuit par jour (avec RH=Repos hebdomadaire).

Semaine	Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
1	M	M	M	RH	RH	S	S
2	S	S	RH	RH	N	N	N
3	N	N	N	N	RH	RH	RH
4	RH	RH	RH	M	M	M	M
5	RH	RH	S	S	S	RH	RH
M	1	1	1	1	1	1	1
S	1	1	1	1	1	1	1
N	1	1	1	1	1	1	1

Figure 2.21 Cycle hebdomadaire de [LNB80]

  Pour un salarié, son planning se lit de gauche à droite et du haut en bas. Arrivé au dimanche de la dernière semaine, il se poursuit au lundi de la première semaine. Un cycle de N semaines est souvent effectué par au moins N salariés (éventuellement +1 pour les congés annuels). Ainsi, pour un jour de la semaine, il y a un salarié sur chaque ligne, donc le total de présence est 1 pour la vacation M, 1 pour S et 1 pour Nuit. Ceci représente la charge hebdomadaire.

- Un cycle mensuel est difficile à concevoir, car chaque mois n'a pas la même durée.

- Un cycle annuel est envisageable, car il y a des contraintes légales pourtant sur l'année (heures annualisées, nombre de jours de congés ou RTT). On peut prévoir des semaines hautes et des semaines basses et utiliser des contraintes sur le volant des heures supplémentaires. Un cycle annuel est un cycle hebdomadaire qui s'exécute en un nombre entier de fois par an, par exemple à 13 (4 fois par an) ou à 17 semaines (3 fois, la dernière semaine se planifie manuellement pour les fêtes de fin d'année).

Un cycle peut être qualifié de rotation en avant (selon la séquence matin – soir – nuit) ou en arrière (nuit – soir – matin). Le premier cycle est très utilisé parce qu'il respecte mieux le biorythme des salariés.

Dans la pratique, des adaptations locales sont applicables par exemple pour traiter une surcharge temporaire [Lap99]. S'il faut un S de plus le samedi, le matin du samedi est retardé de 4 H, et la nuit est avancée de 4 H. S'il faut une matinée M et un Soir S de plus du lundi au vendredi, on peut juxtaposer au cycle de 5 semaines de la précédente figure, un deuxième cycle de 2 semaines :

Semaine	Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
1	M	M	M	M	M	RH	RH
2	S	S	S	S	S	RH	RH
M	1	1	1	1	1	0	0
S	1	1	1	1	1	0	0

Figure 2.22 Cycle hebdomadaire de [Lap99]

2.5.2 Des contraintes sur les cycles

De nombreux travaux portent sur la conception automatique de cycles par ex. [Tré76], [LNB80], [NB92], [Mus00]. Comme le cycle est vécu sur un horizon relativement long, la qualité du cycle doit être travaillée finement. Les conditions suivantes doient être satisfaites d'après [LNB80] :

1. Un changement de vacation se fait seulement après un repos hebdomadaire
2. Le nombre de jours travaillés consécutifs est limité : au moins 2 ou 3 et au plus 6 ou 7 jours.
3. De la même façon, le repos hebdomadaire doit être d'au moins 2 et pas plus de 6 ou 7 jours.
4. Les périodes longues de travail doivent être succédées de longues périodes de repos.
5. Il doit avoir un maximum de week-ends entiers de repos hebdomadaire
6. Les week-ends en repos doivent être *bien* distribués sur le cycle.

D'autres préférences peuvent se rajouter : le travail de nuit ne doit pas être interrompu par des courtes séquences de travail de jour. Des changements fréquents entre le travail de nuit et de jour sont difficiles à assimiler.

[Mus00] relaxe la contrainte 1, mais rajoute d'autres contraintes:

1. Contrainte sur la succession de trois vacations, définie par un tableau à 3 indices.

2. La longueur de chaque suite de vacations identiques doit respecter des bornes minimum et maximum (suivant la vacation)
3. La longueur de chaque suite de vacations de travail doit respecter les seuils

Il est certain que la génération acyclique d'un planning ne respecte pas les mêmes impératifs. Un tel planning ne peut pas servir dans le cadre d'un planning cyclique. Un tel planning sur 12 semaines proposé par Partouche n'avait qu'un seul week-end au repos. Des séquences de 6 jours de travail n'ont pas de périodes de repos allongés.

Semaine	Lundi	Mardi	Mercredi	Jeudi	Vendredi	Samedi	Dimanche
1	S	S	S	S	N	N	R
2	R	M	M	M	M	S	S
3	R	R	N	N	R	R	M
4	M	M	M	M	R	R	M
5	M	M	R	R	S	S	N
6	N	N	R	R	M	M	M
7	M	M	R	R	M	M	M
8	M	R	R	M	M	M	M
9	M	R	R	R	S	S	R
10	R	M	M	M	M	M	M
11	R	R	M	M	M	M	M
12	R	R	S	S	R	**R**	**R**
M	5	5	4	5	6	5	7
S	1	1	2	2	2	3	1
N	1	1	1	1	1	1	1
R	5	5	5	4	3	3	3

Figure 2.23 Planning de 12 semaines proposé dans [Par98]

2.5.3 La génération de cycles

Un processus semblable à celui décrit ci-dessus a été proposé dans [LNB80], [BW90].

1. Estimation de la taille minimale de l'équipe. Ceci donne la longueur du cycle.

   Diverses formules ont été proposées par plusieurs auteurs sous des hypothèses différentes.

2. Définition des configurations admissibles de journées travaillées + journées au repos

3. En tenant compte des bornes sur le nombre de jours travaillés ou de repos, on identifie les périodes de repos admissibles et on génère toutes les possibilités suivant les jours de la semaine.
   On n'en retient que les possibilités qui permettent de satisfaire les besoins annoncés par jour de la semaine. En évaluant chaque configuration, on favorise celles avec week-ends au repos.

4. Affectation des périodes travaillées

[BW90] propose un modèle qui intègre les étapes 3-5 en une seule, permettant de trouver des solutions optimales. Ce modèle définit des nœuds pour chaque jour de l'horizon du cycle. Un arc connectant les nœuds j et k correspond à une période de travail ou de repos, débutant le jour j et terminant le jour k. Pour un problème à M codes horaires, il peut avoir M+1 arcs entre deux nœuds quelconques (y compris le code horaire repos).

Le modèle [Mus00] admet des séquences de vacations contenant plus d'un type de vacation. La solution proposée exploite l'énumération implicite, en prenant soin de générer qu'une seule fois des plannings obtenus en faisant une rotation.

2.5.4 Le déroulement de cycles avec relaxation

Aucun des travaux actuels ne traite l'application d'un cycle dans la pratique. Le cas typique est un déroulement théorique, en ignorant toute absence prévue comme les congés annuels.

Le problème posé est celui de la relaxation pratique d'un cycle dont la définition mathématique est connue parfaitement. Ce sera décrit plus en profondeur au chapitre 4.

Presque 20 ans après sa première communication dans la planification cyclique, Laporte [Lap99] déplore que les méthodes actuelles ne permettent pas de donner des plannings qui tiennent en compte des problèmes de relaxation.

2.6 METHODES DE RECHERCHE STOCHASTIQUE

Un problème de satisfaction de contraintes peut être résolu par des méthodes de recherche arborescente (donc non-déterministe) et stochastique. Ces algorithmes utilisent des heuristiques pour

- Choisir le point de départ
- Sélectionner l'ensemble de points dans l'espace de recherche (le voisinage)
- Choisir le prochain point d'exploration (qui paraît le plus intéressant pour la poursuite de la recherche)

Des méthodes de recherche stochastique sont les algorithmes gloutons, algorithmes génétiques (GA), le recuit simulé ou le tabou. Hormis les GA, ces méthodes maintiennent une seule solution à la fois, et naviguent à petits bonds à travers l'espace de recherche. Un GA peut manipuler plusieurs solutions à la fois et effectue des sauts de portée non-déterminée. Cela donne un caractère non-local aux GA, et le rend plus consistant dans l'obtention des solutions de bonne qualité.

2.6.1 Algorithmes génétiques

Un algorithme génétique GA est un algorithme de recherche locale stochastique qui emprunte le concept de sélection naturelle des espèces pour trouver ainsi des individus le plus apte [Hol76]. Les GA recherchent des solutions optimales, en manipulant des solutions actuelles. Par exemple, ils combinent des éléments de deux bonnes solutions pour en créer une troisième, qui peut représenter un point très éloigné des solutions parents, dans l'espace des solutions.

Méthode de base

i. Le codage d'un chromosome : L'ensemble des solutions retenu par GA sous la forme de *chromosomes,* codés comme un vecteur d'entiers non-négatifs. Chaque chromosome est une collection de blocs (appelés des *gènes)* qui constituent une solution. Le codage informatique des solutions doit faciliter les différentes tâches de l'algorithme génétique.

ii. L'opérateur de CROISEMENT : A partir de deux chromosomes parents et d'un masque constitué d'un vecteur de n éléments $\in$ { 0 |1 }, cet opérateur permet de générer un nouveau chromosome en prenant le gène du premier parent lorsque l'élément correspondant dans le masque est 1 et du second lorsqu'il vaut 0.

iii. L'opérateur de MUTATION : A partir d'un chromosome, on modifie une partie de la solution de façon aléatoire. Par exemple, on supprime un nombre de salariés, à partir des éléments sélectionnés aléatoirement (ex. $X_i = X_i - 1$), puis on augmente le nombre de salariés dans les autres gênes, afin de rendre la solution réalisable en terme de compétences disponibles.

iv. L'algorithme

1. On génère une population initiale de 100 membres, de façon aléatoire.
2. On applique des opérateurs de croisement ou de mutation un nombre déterminé de fois. Les chromosomes résultant sont remis dans la population s'ils présentent un intérêt certain. Ceux qui sont « faibles » seront éjectés de la population.
3. Si le critère d'arrêt n'est pas atteint, alors on réitère l'étape 2.

Problèmes des GA

Epistasies : Le succès de l'application des GA est souvent attribué à la validité de l'hypothèse des blocks de construction[5]. Cette hypothèse stipule que l'opérateur de croisement doit pouvoir combiner des bonnes solutions partielles (ou blocks) pour former des bonnes solutions complètes. Essentiellement l'épistasie est présente si l'aptitude d'une solution n'est pas une combinaison linéaire du taux de l'aptitude de ses éléments. L'épistasie est une mesure de la non-linéarité de la relation entre le codage et la fonction d'aptitude.

Chaque gêne dans un chromosome peut correspondre à une variable dans un CSP. D'après [Lau98], le problème courant est que les différents gènes ne sont pas indépendants. Les contraintes influencent les valeurs simultanées des gènes, ainsi que l'évaluation des chromosomes.

Validité des opérateurs : l'opérateur de croisement est à la base de la genèse de meilleures solutions. Suivant le codage des plannings en blocs, il se peut que le croisement de deux plannings engendre des plannings qui violent des contraintes dures. Par exemple, pour le problème de construction des tours, si un gène représente la suite des affectations de vacations dans un mois pour un salarié, joindre deux bouts de plannings peut résulter en une suite de plannings interdits (ex. Nuit suivi de Matin).

[Eib01] proposent deux méthodes pour résoudre des CSP avec des GA.

- Traiter toutes les contraintes indirectement : la fonction d'aptitude dépend de la violation des contraintes. A chaque itération, la population des solutions devient de plus en plus admissible.

- Traiter toutes les contraintes directement : la violation des contraintes n'affecte pas la fonction aptitude. On y parvient par l'une des approches suivantes :
    - L'élimination des candidats inadmissibles : cette méthode très générale est très peu efficace (elle ressemble à la « génération et test »)
    - Réparation des candidats inadmissibles : lorsqu'elle est possible, cette méthode donne de bons résultats en général, d'après [Eib01].
    - Concevoir des opérateurs qui respectent les contraintes : cette méthode est la plus intéressante mais aussi la plus difficile à réaliser. L'opérateur de mutation paraît bien adapté, car il mélange des chromosomes admissibles des solutions ; le croisement crée de nombreuses violations.

- Traiter une partie des contraintes directement et l'autre indirectement

Diversité des individus dans la population : un GA doit veiller à ce que la population soit suffisamment diversifiée afin de converger vers des bonnes solutions. En conséquence, il faudrait des mesures de similitude ou de diversité. La taille de la population doit être grande.

Résultats de Cai et Li

[5] Building Block Hypothesis

Une méthode GA [BC96] a été utilisée pour résoudre le modèle de Cai et Li [CL00]. Il y a deux compétences, et le personnel est divisé en 3 classes : ceux qui ne possèdent que la première compétence, ceux la seconde compétence et finalement ceux qui possèdent les deux (dont la disponibilité est supposée inférieure à B). Pour la classe de salarié j=1,2,3, soit n_j le nombre de solutions faisables dans l'ensemble S_j. Pour une solution faisable i, on dénote :

X_i le nombre de salariés du premier groupe affecté aux tâches de la compétence 1,
Y_i le nombre du second groupe affecté aux tâches de compétence 2,
Z'_i le nombre du troisième groupe affecté aux tâches de compétence 1,
Z''_i le nombre du troisième groupe affecté aux tâches de compétence 2,

$$S = \{X_1, X_2, \ldots, X_{n1}, Y_1, Y_2, \ldots, Y_{n2}, Z'_1, Z'_2, \ldots, Z'_{n3}, Z''_1, Z''_2, \ldots, Z''_{n3}\}$$

où n = n1+n2+2n3, le nombre total de solutions, X,Y,Z' sont des gênes. Chaque élément du vecteur représente une des solutions faisables pour chaque type de personnel.

Utilisés dans le contexte de personnel à multiples compétences, sur un horizon d'une semaine, des intervalles d'une heure. Il y a des vacations de nuit (23H-07H) et de jour : 7H-16H, 8H-17H, 9H-18H, 10H-19H, 11H-20H, 12H-21H, 13H-22H, 14H-23H, 15H-0H. La durée des pauses pendant les vacations de jour, est d'une heure après 4 heures de travail. Les horaires hebdomadaires doivent avoir au maximum 3 nuits consécutives, avec une journée de repos compensateur et suivi du repos hebdomadaire. Les vacations journalières sont maximums de 6 jours de suite, suivies du repos hebdomadaire (un jour quelconque de la semaine). Pour chaque salarié, une semaine jour commence toujours à la même heure. Sur une station Sun SPARC, l'implémentation en C prend 8 à 10 minutes pour produire des solutions faisables, avec un total de 593 salariés.

Conclusion

Par rapport aux autres méthodes qui recherchent un chemin à la fois, en considérant plusieurs solutions simultanées le GA est susceptible de trouver des solutions plus intéressantes. Pour la génération de la population initiale de solutions, il faut utiliser des méthodes supplémentaires, par exemple une méthode gloutonne.

Il est difficile de s'assurer qu'à partir de solutions consistantes, les opérateurs produisent des solutions filles consistantes. Plus les contraintes sont complexes, plus l'élimination des solutions inconsistantes prend du temps.

2.6.2 Algorithmes mimétiques

[Daw76] a introduit l'idée de mime comme une alternative au concept de gène. Non seulement le matériel génétique est transmis d'une génération à la suivante, mais aussi des idées et concepts. Le concept de mime est assimilé comme une unité d'information passée entre des individus. Quand un gène est passé aux descendants, il n'est pas altéré : les individus n'ont aucun contrôle sur leur composition génétique. Par contre, l'individu peut adapter le mime à son propre environnement.

La notion d'un algorithme mimétique a été introduit dans [MN91] pour décrire des algorithmes incorporant la recherche locale (par ex. hill climbing) suite à une mutation. Afin de réduire le temps d'évaluation, les algorithmes exploitent la différence entre les solutions finale et initiale, par ex. [Kra96].

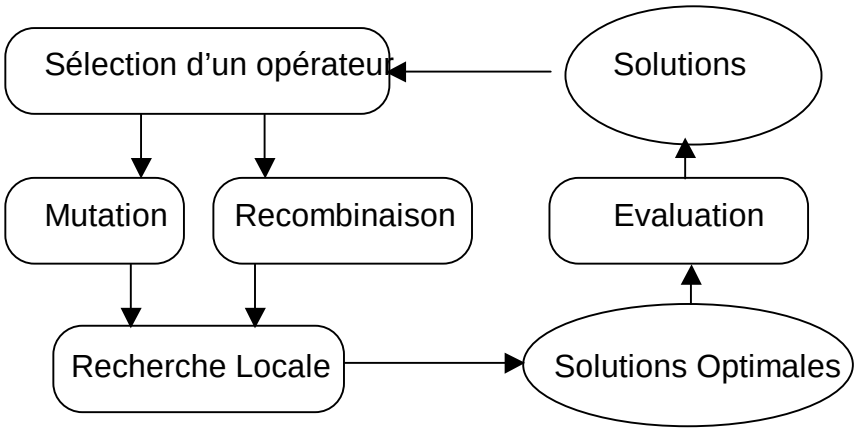


Figure 2.24 Le schéma global d'un algorithme mimétique[6]

2.6.3 Le recuit simulé

Le processus du recuit est le lent refroidissement d'un solide. Le premier algorithme permettant de simuler ce processus a été proposé initialement par [MRT53]. Plus tard, Kirkpatrick *et al.* établissent l'équivalence entre le processus de recuit et la résolution d'un problème d'optimisation combinatoire. L'équivalence est obtenue en faisant une analogie entre l'état des particules et les solutions d'une part, et entre l'état d'énergie et le coût d'autre part.

La méthode de Metropolis génère une séquence d'états qui illustre l'évolution du corps de l'état liquide à l'état solide, en observant l'équilibre thermique à chaque instant. Le prochain état est obtenu par une petite perturbation de l'état courant, générée de façon aléatoire, et un critère d'acceptation. Ainsi une perturbation n'est pas acceptée systématiquement. Cela dépend de la différence d'énergie entre l'état courant et le nouvel état, et la température ambiante.

Une configuration est une solution et une transition de i à j représente une perturbation en configuration i, la transformant en configuration j. Soit Fi et Fj le coût des configurations i

[6] Burke E.K. , Newall J.P. , Weare R.F. , A mimetic algorithm for university exam timetabling, in The Practice and Theory of Automated Timetabling: selected papers from the 1st Int'l. Conf. on the Practice and Theory of Automated Timetabling, Napier University. Lecture Notes in Computer Science Series, Vol. 1153, pp. 241-250, 1995.

et j, et T représente la température. La probabilité d'accepter la transition de i vers j est définie par le critère d'acceptation de Metropolis :

$$A_{ij}(T) = 1 \quad \text{si } F_i \geq F_j \qquad (8)$$
$$= e^{(Fi - Fj)/T} \quad \text{sinon, avec } T > 0$$

L'algorithme RS classique est une séquence des algorithmes Metropolis qui est évaluée pour une suite décroissante de température. Par conséquent, les caractéristiques globales du RS est que la probabilité d'acceptation des transitions de coût croissant, décroît avec la baisse de température. Quand la température tend vers zéro, la détérioration du coût n'est plus acceptée.

La nature stochastique du RS est décrite par le critère d'acceptation et une fonction de voisinage. Ce dernier définit pour chaque configuration i, l'ensemble de configurations voisines N(i) qui peuvent être atteintes avec une seule transition. Ainsi de deux choses l'une : a) les configurations qui ne sont pas voisines auront une probabilité nulle de devenir la prochaine configuration ; b) les configurations qui sont voisines auront une probabilité non nulle de succéder la configuration courante.

Algorithme du recuit simulé (α, T°, N, P)

```
α = taux de refroidissement entre 0,5 et 0,99
T° = température initiale
N = nombre de paliers de températures
P = nombre d'étapes par palier

Soit
    S = Solution initiale générée de façon aléatoire
    T = T°
    Tant que i < N faire
        J = 0
        Tant que J < P faire  'Une étape
            S' = Générer une transition à partir de S
            Δ E = E_S' – E_S
            Si Δ E ≤ 0 Alors
                S = S' // Accepter la transition
            Sinon
                Proba = exp^(– Δ E/T)
                Si aléatoire(0,1) < Proba Alors
                    S = S'  // Accepter la solution
                Sinon
                    // Rejeter la solution
                Fin Si
            Fin Si
        Fin
        T = α * T
    Fin
    Solution finale = S
```

Figure 2.25 L'algorithme du recuit simulé

Le générateur de nombres aléatoires entre 0 et 1 utilise une loi de distribution uniforme.

2.6.4 Recherche Locale

Méthode d'échanges locaux

[JBS94] propose un algorithme simple qui construit un ensemble de tours et améliore la qualité en faisant des échanges. Pour chaque jour de l'horizon, les vacations sont triées par ordre croissant de l'heure de début. Un ensemble de tours est généré en tenant compte des repos hebdomadaires. Pour chaque tour, calculer la variance de l'heure début des vacations.

Soit i* le tour dont la variance est maximum sur l'ensemble des tours. Considérons l'ensemble des tours avec au moins une vacation commençant plus tôt que la vacation débutant le plus tard dans i* et avec une vacation commençant plus tard que la vacation débutant le plus tôt dans i*. Pour chacun tour j, identifier la séquence d'échange de vacation permettant de réduire la somme des variances de i* et j. Répéter cette étape un nombre donné de fois, tant que la variance de i* continue à diminuer.

Des variantes de ces méthodes réduisent la différence en heure de début des vacations dans un tour.

Méthode de recherche locale avec redémarrage aléatoire

Les méthodes de recherche locale ont la tendance d'être piégées par des optima locaux. [MK01] propose une méthode permettant de sortir de cette impasse : lorsque le nombre de retour-arrière dépasse un seuil préétabli, la recherche systématique est arrêtée autour de la solution partielle courante, un autre point de l'espace de solutions est généré de façon aléatoire et la recherche locale est redémarrée. Inspiré de [GK98] et incorporant des techniques de *forward-checking* et vérification des consistances, une amélioration de l'ordre d'une grandeur a été rapportée dans [MK01].

2.6.5 Recherche Tabou[7]

Initialement proposé par F. Glover [Glo89], la recherche tabou est un méta-heuristique de recherche locale à partir d'une *solution initiale*, dans un *voisinage prédéfini*. L'algorithme parcourt le voisinage de l'espace de solutions depuis la solution initiale, de façon non-ordonnée. Le voisinage est défini par un ensemble de types de transitions d'état. On ne retient que la solution qui possède une évaluation meilleure à celle obtenue auparavant.

Afin d'éviter de revenir sur une solution déjà visitée, le système maintient une liste de transitions récentes qui sont déclarées interdites, d'où le nom *tabou*. Cette liste fonctionne comme une pile, les éléments seront tabou pendant un nombre K d'itérations. Ainsi, faute de trouver une solution avec une meilleure évaluation, on peut accepter en une qui n'est pas sur la liste tabou.

Cette technique dispose d'un mécanisme permettant de retenir une solution malgré le tabou : un *critère d'aspiration* souvent utilisé est basé sur le taux d'amélioration de la fonction d'évaluation. Si la solution est « très » bonne par rapport à la solution courante, on l'accepte malgré son inscription sur la liste tabou.

[7] Taboo Search

Un *mécanisme de diversification* est nécessaire pour réduire la possibilité d'être piégé dans un optimum local.

Algorithme de recherche tabou (f, x*)

Entrée : Solution admissible x*
Fonction d'évaluation z* = f(x*)
Résultat : x* est la meilleure solution avec la valeur z*

Soit la solution à l'étape courante k : $x^k := x^*$
Initialiser la liste tabou L ← ∅
Tant que Critère d'arrêt insatisfait
Début
(1) Recherche dans le voisinage de la solution courante
Soit $V(x^k)$ = l'ensemble des solutions voisines de x^k,
obtenues avec les transitions admissibles à partir de x^k
Soit $V_L(x^k) \subseteq V(x^k)$ obtenu en éliminant les solutions interdites par la liste tabou

Certaines solutions ne sont pas éliminées si le critère d'aspiration est satisfait
par exemple si son évaluation est très intéressante par rapport à z*

Choisir la solution $x° \in V_L(x^k)$ qui minimise la fonction f
Si f(x°) < f(x*) alors faire $x^{k+1} \leftarrow x°$

(2) Gestion de la liste tabou L
Si la liste a atteint la longueur maximale, éliminer l'élément la plus ancienne
Ajouter dans L l'information sur la transition $x^k \rightarrow x^{k+1}$.
Fin

Figure 2.26 L'algorithme de recherche tabou

Pour résoudre le problème de l'emploi du temps, [SS95] utilise le codage matriciel $M_{i,k}$ qui contient le nom de la classe du professeur j à la période k. Les voisinages proposés sont :

- Echanger deux cours pour un même professeur
- Déplacer un cours à une autre période

[CST00] utilise une liste tabou de taille variable. A son entrée dans la liste, à chaque solution est attribué un nombre aléatoire (choisi entre deux valeurs fixes K_{min} et K_{max}) celui ci représente le nombre d'itérations la solution doit rester dans la liste. Le voisinage est défini par le remplacement d'un employé i à la période j effectuant la tâche k, par un autre employé j qui ne travaille pas à la période j.

Problèmes de la recherche tabou

Comme les méta-heuristiques, la recherche tabou est une technique d'optimisation sans contraintes. Effectivement, les transitions d'un état à un autre peuvent engendrer des violations de contraintes, sauf si elles ont été conçues spécifiquement.

2.6.6 Greedy Randomized Adaptive Search Procedures

Proposé la première fois dans [FR95], le méta-heuristique **GRASP** consiste en deux étapes à chaque itération :

- La construction d'une solution initiale avec une fonction gloutonne aléatoire *adaptative*. La fonction gloutonne prend en compte les choix fait précédemment pour déterminer la prochaine solution.
- L'utilisation d'une procédure de recherche locale pour l'améliorer.

La meilleure solution sur toutes les itérations sera retenue comme solution finale. Bien évidemment, on peut tomber dans un optimum local dans une itération, mais le redémarrage aléatoire permet d'explorer l'espace de solutions pour en trouver une qui soit proche de l'optimum global.

Algorithme GRASP[8](MaxIterations, Seed)

Nombre maximal d'itérations est donné en entrée
Seed permet d'initialiser le générateur de nombres aléatoires

```
Solution ← 0, MeilleurSolution ← 0
Pour K=1, Max_Iterations Faire
    Solution ← GreedyRandomizedConstruction(Seed)
    Solution ← local_Search(Solution)
    MiseAJour (Solution, MeilleurSolution)
Retourner MeilleurSolution
Fin

Procédure GreedyRandomizedConstruction(Seed)
Solution ← 0
Initialiser les générateurs de nombres aléatoires avec Seed
Evaluer la solution
    Tant que Solution est incomplète
        Construire une liste de candidats (de façon adaptative à la solution partielle)
        Sélectionner aléatoirement un élément s de la liste de candidats
        Solution ← Solution ∪ {s}
        Réévaluer le coût incrémental
    Fin Tant que
    Retourner Solution
Fin procédure
```

Figure 2.27 L'algorithme GRASP d'après [RR01]

Dans l'application de cette méthode au problème de l'affectation généralisée[9], la probabilité du choix d'un candidat est en fonction du rapport b_j/a_{ij}, b_j étant la capacité maximale de l'agent j, et a_{ij} la consommation en ressource j par la tâche i si affectée à la ressource. Ce rapport est grand si la tâche i n'utilise qu'une petite partie de la capacité de la ressource j. Le candidat est admis si la contrainte de capacité n'est pas transgressée. Sinon, le premier candidat avec une capacité restante suffisante est pris. Si aucun candidat n'a de la capacité restante, on sélectionne avec une probabilité uniforme. Nous voyons que la méthode est s'adapte aux choix précédents dans la solution courante, et ignore les solutions précédentes.

Au total, des centaines voire des milliers d'itérations peuvent être exécutées, chaque démarrage exploite un point pris de façon aléatoire. Cette technique est conceptuellement

[8] Une liste d'applications GRASP est visible à l'adresse http://www.research.att.com/~mgcr, qui est géré par Rescende.
[9] Generalized Assignment Problem (GAP)

facile à déployer sur un grand nombre de machines en parallèle. Comme on dispose d’une solution réalisable à chaque itération, on peut arrêter le processus lorsqu’on a besoin d’une solution.

2.6.7 Conclusion sur les méthodes stochastiques

En général l’application de ces méthodes à beaucoup de problèmes de taille réelle montre qu’elles donnent des solutions de bonne qualité dans de nombreux cas. Nous pouvons faire deux remarques à propos de ces méthodes.

D’une part, n’étant pas basées sur le raisonnement logique, elles seraient difficiles à appliquer à la résolution des modèles traitant plus de types de contraintes.

D’autre part, ses résultats semblent découler du grand nombre de calculs de solutions et d’évaluation. Typiquement dans la littérature de recherche sur les GA, on parle de 500 ou 1000 itérations. Donc, elles exigent un temps potentiellement inacceptable pour un produit commercial.

2.7 LES ALGORITHMES D'APPROXIMATION

La théorie de la complexité (Cook en 1971 et Karp en 1972), postule que tout problème NP complet pourra être résolu dans un temps polynomial en fonction de la taille du problème, si et seulement si tous les autres problèmes NP complets peuvent être résolus dans un temps polynomial. La classe de problèmes NP complets inclut des problèmes pratiques tels que le voyageur de commerce, couverture d'ensemble, programmation en nombres entiers avec des variables limitées. Après des décennies de recherche algorithmique, des générations de chercheurs n'ont pas trouvé de tels algorithmes. Par contre, ils ont trouvé de plus en plus de problèmes de la même classe.

En conséquence, de nombreux chercheurs [Sah77], [Joh74] ont proposé des algorithmes pour résoudre les problèmes NP complets de façon presque optimale : ces algorithmes sont capables de résoudre ces problèmes dont l'évaluation est proche de la valeur optimale. Cette tendance est renforcée par le fait que les problèmes sont souvent définis de manière approximative par rapport à la réalité.
Définition : Un algorithme est appelé algorithme ε-approché pour un problème P si pour toute instance I du problème, F*(I) >0 l'évaluation d'une solution optimale à I, F°(I) >0 est la valeur générée par l'algorithme, et ε est une constante, alors l'erreur relative est :

$$\frac{|F^*(I) - F^\circ(I)|}{F^\circ(I)} \leq \varepsilon \qquad (9)$$

Pour un problème de maximisation, $0 \leq \varepsilon < 1$ et un problème de minimalisation $\varepsilon \geq 1$.

Définition : un schéma d'approximation en temps polynomial PTAS [10] pour un problème de minimisation est une famille d'algorithmes $\{A_\varepsilon ; \varepsilon > 0\}$, tel que pour chaque ε , A_ε est un algorithme $(1+\varepsilon)$ -approché avec une complexité en ε. Pour un problème de maximisation, A_ε est un algorithme $(1-\varepsilon)$ -approché

Sans entrer dans les détails de cette branche d'étude[11], il existe des PTAS pour les problèmes tel que sac à dos, voyageur de commerce sur un plan Euclidien, certains problèmes d'ordonnancement et la couverture d'ensemble.

2.7.1 La couverture d'ensemble[12]

Etant donné un ensemble U d'éléments, une famille F de sous ensembles de U : S_1 , S_2, …, $S_n \subseteq U$, avec des poids associés w_1 , w_2, …, w_n, le problème de couverture de U consiste à trouver l'ensemble $I \subseteq \{1, 2, \ldots, n\}$, dont l'union recouvre les éléments de U.

$$U = \cup_{i \in I} S_i \qquad (10)$$

qui minimise la somme $\sum_{i=1,n} w_i$..

Il y a deux méthodes possibles dans l'énumération de toutes les couvertures I. L'une consiste à construire toutes les collections, et stopper la recherche dès que l'union recouvre U. Cette méthode est qualifiée de l'énumération des sous-ensembles. L'autre

[10] Polynomial time approximation schema PTAS
[11] le lecteur pourra consulter le tutorial de 132 pages [Wil98]
[12] Set Covering Problem

consiste à augmenter le taux de couverture à chaque étape ; la méthode est qualifiée de l'augmentation de la couverture.

L'un des premiers **algorithmes d'approximation** pour résoudre ce problème a été proposé par [Joh74] avec une complexité de O(n log n). A chaque étape, la technique glouton consiste à ajouter le sous-ensemble S_i qui contient le plus d'éléments non encore couverts.

La proposition de Chvàtal [Chv79] donne une complexité de O(m log n) où $m = \sum_{i=1,n} | S_i |$, le résultat a le poids W qui satisfait la borne supérieure :

$$W \leq W\ opt\ .\ O(\log d), \text{ avec } d = Max\ |\ Si\ | \qquad (\mathbf{11})$$

Ce résultat a été amélioré dans [Hoc82] donnant le poids W

$$W \leq W_{opt}\ .\ f,$$

avec f = le plus grand nombre d'ensembles S_i qui contiennent un élément x quelconque

Soit $F_j = \{I \mid e_j \in S_i\}$, alors $f = Max\ |F_j|$

2.7.2 L'algorithme Yehuda et Even

L'algorithme de [BE81] obtient le même résultat, mais évite l'appel à un algorithme de Programmation Linéaire. Sa complexité est proportionnelle à $m = \sum_{i=1,n} |S(i)|$.

Algorithme de Behuya et Even ()

Famille de n ensembles $F = \{S_1, \ldots S_n\}$, poids de chaque $S_i = w_i$ nombre real ≥ 0
L'ensemble universel $U = \{e_1, \ldots e_t\} = U_{i=1,n} S_i$, t = nombre d'éléments distincts

Traitement Soit $N = \{1, \ldots, n\}$, $T = \{1, \ldots, t\}$, $F(j) = \{ i \mid e_j \in S_i \} \subseteq N$

Initialisation : $\forall I \in N$, $SW(I) = w_i$
I = 0, J = T
Itération :
Tant que ($J \neq 0$) et soit $j \in J$
Choix d'un S_i :
M = min { SW(I) | $I \in F(j)$ }
Soit i=k pour SW(k) = M
Mise à jour des compteurs
$\forall I \in F(j)$, SW(I) = SW(I) – M
I = I u {k} – choix du sous-ensemble k
$J = J \setminus S_k$ – enlève toutes les périodes couvertes par cet ensemble
Fin

Sortie : $I \subseteq N$, contenant les indices de S choisis, et qui minimise $\Sigma_{i \in I}\ w_i$

Figure 2.28 L'algorithme de Behuya et Even

Une comparaison de 9 algorithmes pour résoudre le problème de couverture d'ensemble est proposée dans [GW97]. Les meilleurs résultats ont été trouvés par les méthodes suivantes :

- Algorithme Glouton (Chvàtal) : A chaque itération, on choisit la variable qui apparaît dans le plus grand nombre de contraintes non satisfaites. En cas d'égalité, la variable avec le plus petit indice est choisie.

- Algorithme Glouton-Aléatoire : A chaque itération, on choisit la variable qui procure le plus de gain. En cas d'égalité, on utilise une règle aléatoire pour choisir parmi les candidats. Les calculs sont répétés N=100 fois et la meilleure solution est retenue. Cet algorithme donne les meilleurs résultats dans la comparaison (presque toujours la solution optimale), mais la consommation en temps machine est importante.

2.7.3 Multiples couvertures d'ensemble[13]

La couverture d'ensemble ne permet que de couvrir une seule fois l'ensemble U. La construction de vacations exige la multiples couvertures car à un instant donné, les besoins en personnel sont en général supérieurs à 1.

[PSW97] analyse une généralisation du problème de couverture d'ensemble avec la classe de problèmes dénotée par ILP(k, b). Cela consiste à trouver le vecteur $\mathbf{x} \in \{0,1\}^n$ en minimisant $\sum_j x_j$, soumis aux contraintes $\mathbf{A}\,\mathbf{x} \geq \mathbf{b}$, où **A** est une matrice booléenne m x n avec au plus k fois 1 par rangée, **b** est un vecteur en nombres entiers et b est la plus petite valeur dans **b**. Lorsque k=n, b=1, on retrouve une variante du problème de couverture d'ensemble. Des algorithmes d'approximation avec un rapport $\varepsilon = (k – b + 1)$ ont été proposés, voir [HH86] par exemple.

[PSW97] propose un nouvel algorithme ε-approché pour résoudre ILP(k, b) où

$$\varepsilon = (k – b + 1) (1 – (c/m)^{d}) \qquad (12)$$

avec une petite constante c > 0, d = 1/ (k-b+1), m = nombre de rangées de la matrice **A**. L'algorithme proposé utilise la programmation linéaire comme étape intermédiaire.

2.7.4 Conclusion

Cette revue des méthodes de résolution du problème de couverture d'ensemble, permet de donner une idée des résultats obtenus dans ce domaine. La construction des vacations relève du problème à multiples couvertures d'ensemble. U est l'ensemble des intervalles de temps de la journée ; S_i sont les horaires légaux, compte tenu de la législation sur les durées de travail et de repos. La couverture de l'ensemble U traduit la présence de personnel sur la journée.

Cependant, il y a d'autres contraintes non-couvertes par ce problème :

- La multiple compétence : dans un CALL CENTER, le personnel est souvent compétent pour traiter plusieurs flux d'appels.

- Le temps partiel de certains employés

[13] Multiple Set Covering

- Les préférences individuelles

Compte tenu de la complexité de ces algorithmes, il n’est pas envisageable d’enrichir le modèle de planification, de tenir compte des préférences ou de l’historique.

2.8 LES SYSTEMES INTERACTIFS D'AIDE A LA DECISION

Cette thèse a pour finalité la création d'un système logiciel pour la création de plannings. Jusqu'à ce point, nous avons analysé les méthodes techniques pour générer automatiquement des plannings. Or la mise en œuvre de ce type de logiciel n'est pas aussi simple que celle des traitements de textes.

Ce paragraplhe propose d'explorer les modèles conceptuels d'un système logiciel, puis d'un système d'aide interactive à la décision, et enfin d'un logiciel planning.

2.8.1 Le modèle conceptuel d'un système logiciel

Un modèle conceptuel **MC** est une représentation mentale du fonctionnement du système et il indique comment les contrôles de l'interface utilisateur le modifient.

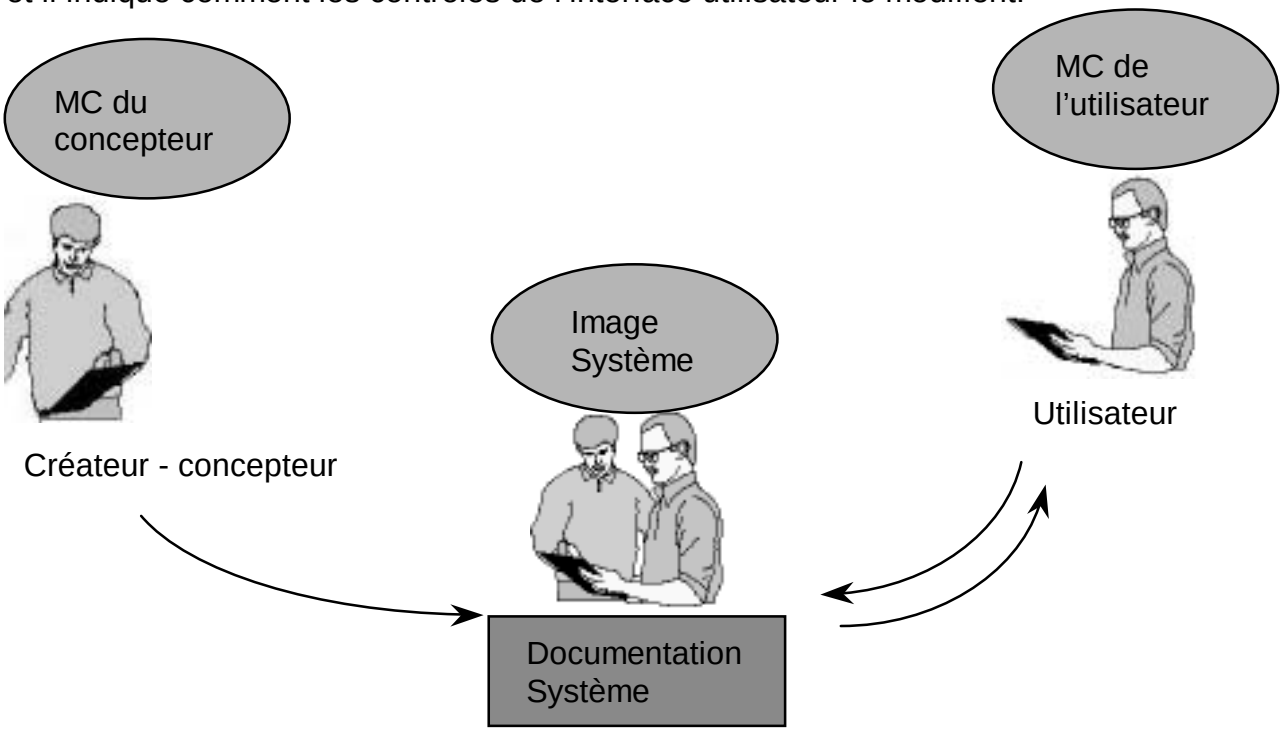


Figure 2.29 Les différents types de modèles conceptuels d'après Norman

Dans la science de communication, [Nor90] proposa trois types de modèles :

1. Modèle concepteur : un modèle conceptuel du créateur du système dans sa globalité.
2. Image Système : la représentation physique externe du système, ce que voit, entend ou ressent tout utilisateur qui interagit avec le système. Le créateur communique à l'utilisateur via l'image système.
3. Modèle utilisateur : un modèle conceptuel de l'utilisateur du système. Il le construit à partir de son expérience, des documents pédagogiques et de la formation qu'il a reçue.

Si l'utilisateur dispose d'un bon modèle conceptuel, il saura prédire les effets de ses actions, donc capable de réussir l'interaction avec le logiciel.

Afin que l’utilisateur puisse exploiter effectivement tout système, Norman propose que le modèle conceptuel de l’utilisateur soit proche de celui du concepteur [Nor90]. Je cite : "Important image is not the one on the screen but in the users' mind”. L’utilisateur crée son modèle conceptuel à partir de son expérience, en lisant la documentation système, ou en suivant les cours. Si les modèles conceptuels de l’utilisateur et du concepteur ne coïncident pas, l’utilisateur fera de nombreuses erreurs et rencontrera des problèmes chaque fois qu’il se servira du logiciel. Il aura besoin de beaucoup de temps pour maîtriser le logiciel et sera frustré.

Un système réussi doit donner des indices visuels quant à leur fonction ou opération. Ils doivent être justes (sans exagération) et proportionnés (liés aux tâches de haut niveau et non des détails d’implantation). L’idéal est de créer des métaphores ou modèles conceptuels.

Une métaphore nous permet de voir des correspondances entre deux choses apparemment désassociées. Elle nous invite à entrevoir deux choses sous un nouvel angle. Pour Aristote le grand philosophe de la Grèce Antique, devenir un maître de métaphore est un but en soi : c’est une capacité qui ne peut s’apprendre des autres, un signe de génie, car une bonne métaphore nécessite une perception intuitive des analogies parmi des phénomènes disparates[14].

Avec des systèmes de plus en plus complexes, l’utilisation d’une seule métaphore peut atteindre ses limites. Une métaphore ne pourrait pas représenter tous les aspects du comportement du système. Plusieurs métaphores peuvent être utilisées conjointement. L’utilisateur doit être encouragé à utiliser des parties relevants de chaque métaphore. Cependant, il convient d’être attentif car de fausses interprétations de la métaphore peuvent engendre des erreurs.

[14] By far the greatest thing is to be a master of metaphor. It is the one thing that cannot be learned from others. It is a sign of genius, for a good metaphor implies intuitive perception of similarity among the dissimilar.

2.8.2 Les principes d'un SIAD

Un SIAD est un outil informatique qui assiste le décideur tout au long de sa prise de décision. En effet un processus de décision n'est pas entièrement automatisable [Pom92]. Il est nécessaire de garder l'utilisateur dans le processus de décision car il est capable de considérer des critères qui ne peuvent pas être explicitement modélisés dans un système. L'identification des points du processus qui peuvent être automatisés (ou traités automatiquement) constitue une étape importante dans la construction d'un SIAD. Un processus de décision dans la gestion des organisations se compose de quatre phases : [LP89]

- Phase d'information
- Phase de conception
- Phase de choix
- Phase d'évaluation du choix

Le déroulement des phases n'est pas forcément séquentiel : certaines peuvent se dérouler en parallèle. Des retours en arrière peuvent se produire notamment lors de la phase de conception ; l 'élaboration d'un scénario peut nécessiter l'acquisition d'informations supplémentaires.

Si la phase de choix relève du décideur seul, un système d'information a sa place dans les deux phases de préparation à la prise de décision et dans l'évaluation du choix. En effet, la capacité de traitement des informations des ordinateurs permet au décideur, pendant la phase d'information, d'accéder rapidement à des informations brutes ou traitées concernant la situation courante et le champ des manœuvres autorisées par exemple. Cette capacité de traitement de l'information peut être aussi utilisée lors de la phase de conception. Dans cette phase, le système d'information peut fournir des éléments d'évaluation des scénarii à l'aide d'indicateurs calculés à partir de modèles ou de procédures de calcul adaptées. La phase d'évaluation du choix correspond à une évaluation a posteriori du choix du décideur ; cette évaluation permet de corriger les petites erreurs. La détection des erreurs et des aspects à améliorer peuvent être facilités par l'apport d'informations et d'indices calculés par le système d'information [Lév89].

L'association d'un système informatique et d'un décideur permet une symbiose complémentaire. Le décideur de par sa connaissance pratique et son expérience possède un méta-modèle du processus de décision. Le SIAD, par sa capacité de traitement de l'information, l'aide à structurer le modèle. Le décideur contrôle le processus de décision et le SIAD l'assiste en effectuant les calculs standards et répétitifs sur les données. Le processus de décision s'apparente à une recherche heuristique menée par le décideur ; le système jalonne le processus de recherche à l'aide d'indicateurs et d'informations. En fonction de ces informations, le décideur continue l'exploration heuristique des actions possibles ou s'arrête si tout indique que la solution construite rencontre ses buts de façon satisfaisante [LP89].

2.8.3 Le modèle conceptuel d'un SIAD

Un SIAD se compose de trois modules : un module de dialogue, un module de données et un module des procédures de calcul ou modèles. Le module de dialogue est connecté aux deux autres modules, et constitue l'interface entre l'utilisateur et le reste du système.

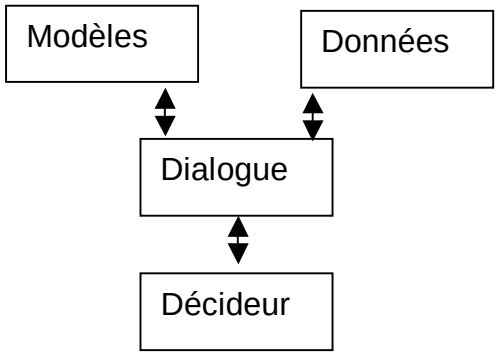


Figure 2.30 Modèle conceptuel d'un SIAD

Le module de dialogue permet au décideur d'accéder aux données et aux modèles du SIAD. Ce dernier utilise ce même module pour communiquer le résultat des manipulations effectuées par le décideur. Les échanges sont d'autant plus favorisés que les représentations des résultats, tout comme le mode de questionnement du système, correspondent aux représentations mentales du décideur. Ainsi, le décideur peut exercer son contrôle et effectuer sa recherche heuristique dans de bonnes conditions.

Le module de données assure la fonction de mémoire : il stocke non seulement les données, de façon permanente (persistante) ou passagère, mais il gère aussi l'enregistrement de données volatiles ainsi que l'effacement de ces mêmes données selon le souhait de l'utilisateur. Ces données volatiles correspondent aux résultats obtenus au cours de traitements de données. Les données que nous avons qualifiées de permanentes sont les statistiques ou autres données qui décrivent la situation courante et passée. Parmi ces données, il peut aussi y avoir des estimations concernant l'évolution de certains paramètres environnementaux.

Le module de modèles contient l'ensemble des procédures de calculs utilisées dans les différents traitements mis à la disposition de l'utilisateur. Il peut s'agir des calculs standards et de procédures de représentation de données.

2.8.4 Modèles conceptuels d'un logiciel de planning

Pour mémoire, nous rappelons plusieurs modèles conceptuels souvent utilisés pour les plannings muraux. Ces plannings sont des tableaux, dont un axe est l'axe du temps.

- Plannings par tâche : le diagramme de Gantt où le temps occupe l'axe horizontal et une ligne représente typiquement une tâche. Dans des projets avec beaucoup de tâches, on les regroupe en projets et sous projets. La durée totale d'un projet est la durée cumulée des tâches sur le chemin critique du projet.

  On peut y représenter les dates de début et de fin des tâches, ainsi que les relations de précédence entre elles.

  Ce planning permet de suivre l'achèvement d'un projet et déterminer le retard qu'une tâche peut subir avant que le projet ne soit pas affecté.

  Par contre il ne permet pas de suivre les activités de chaque ressource.

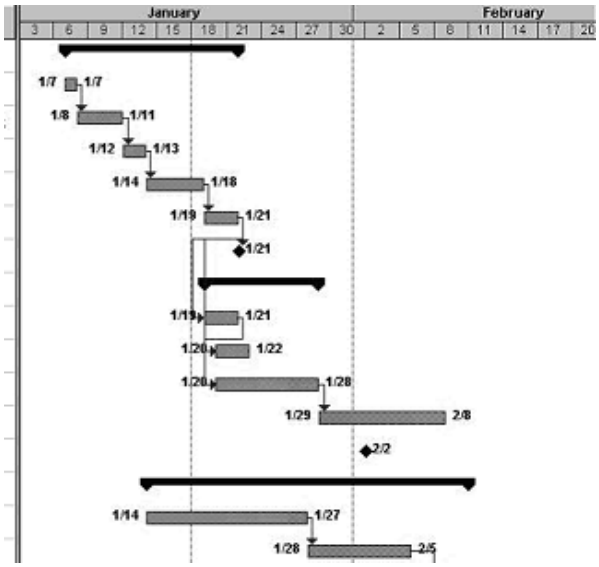


Figure 2.31 Un diagramme de Gantt

- Plannings par ressource : le temps occupe l'axe horizontal et chaque ligne représente le planning de la ressource. Ce dernier peut être une personne, une machine ou une salle.

- Plannings par poste de travail : le temps occupe l'axe horizontal et chaque ligne représente le planning du poste.

2.9 COURBE DE CHARGE ET DIMENSIONNEMENT

La phase de calcul de charge sert à fournir des données sur la charge de travail prévue sur chaque intervalle de temps. Comme on en déduit les besoins en personnel, elle est nécessaire à toute planification des horaires du personnel.

Ensuite, il faut calculer la taille des équipes capables de faire face aux charges. Nous rappelons les travaux courants permettant de dimensionner les équipes.

Comme cela se fait dans l'esprit de la planification en gestion de production, nous poursuivons notre étude sur l'application du dimensionnement à des fins de planification du personnel.

2.9.1 La courbe de charge

La planification des horaires du personnel prend comme entrée le besoin en nombre de personnes pour assurer un niveau de service donné. Or fréquemment, la donnée disponible est la quantité de travail ou service pendant chaque intervalle de temps. Pour retrouver le besoin en nombre de personnes, une loi linéaire ou règle de trois est souvent utilisée.

Dans les centres d'appel, [Seg74] rappelle la notion du *niveau de service* dans la forme « pas plus de α% des clients en attente de plus de β unités de temps ». En supposant qu'un état d'équilibre statistique est atteint au cours de chaque intervalle de temps, avec une arrivée de clients suivant une loi de Poisson avec le taux λ, les temps d'attente suivent une distribution exponentielle décroissante avec une moyenne de la durée de traitement $1/\mu$ donnant une charge $a = \lambda/\mu$. Le nombre d'opérateurs s est le plus petit entier tel que :

$$\alpha \geq C(s,a)\, e^{-\beta\mu(s-a)} \qquad (13)$$

où C est la loi d'Erlang C

$$C(s,a)=\frac{(a^s/s!)*(s/(s-a))}{\sum_{k=0}^{s-1} a^k/k! + (s/(s-a))} \qquad (14)$$

Les besoins en personnel pour chaque intervalle de temps, relèvent souvent des statistiques. Il est généralement traité par une méthode probabiliste, alimentée par les historiques des demandes (de nature probabiliste), éventuellement corrigée par la présence des événements exceptionnels sur la période à étudier. Une distribution probabiliste p peut représenter le nombre de salariés r dont on a besoin.

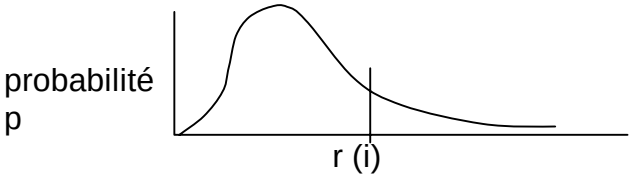


Figure 2.32. Besoins dans l'intervalle i considéré

Le décideur doit aussi estimer l'absentéisme et augmenter la demande pour pouvoir y faire face. Sinon, il faut prévoir d'autres méthodes d'organisation pour le traiter (équipes volantes, temps partiels). Le décideur doit décider quelle proportion de cette demande à traiter effectivement.

2.9.2 Le dimensionnement de l'équipe

Avant de pouvoir planifier l'emploi du temps d'une équipe, il est nécessaire de connaître sa taille. Dans la littérature scientifique, de nombreux auteurs proposent des méthodes de calcul de la borne inférieure avec des règles de travail contraignantes. Dans le même temps, ils proposent des algorithmes permettant de faire des plannings sur un nombre réduit de semaines ; ces plannings peuvent alors être utilisés en plannings cycliques. Les algorithmes sont très rapides en exécution car ils exploitent les structures particulières du problème et les résultats sont de bonne qualité. Dans la suite du paragraphe, seules les bornes inférieures de la taille de l'équipe sont présentées.

a. Cas d'une seule vacation (demandes par jour de la semaine)

Burns et Carter ont été les premiers à proposer une méthodologie basée sur le calcul de plusieurs bornes inférieures, la taille étant le maximum de ces bornes M [BC85]. Ils ont considéré le cas d'une seule vacation par jour et le besoin est n(d), d=1 pour le dimanche, …, 7 pour le samedi. W = nombre total de salariés, n = Max(n(1), …, n(7)).

Méthode de Burns et Carter

La taille d'une équipe dépend de plusieurs facteurs :

1. Charge de week-end : le nombre de salariés disponible en moyenne chaque week-end doit être suffisant pour respecter les besoins du week-end.

   Sur B semaines, chaque salarié est disponible (B-A) week-end. (B-A)W= nombre de personne-jour disponible pour le week-end, Bn = maximum des besoins de week-end. (B-A)W ≥ Bn, d'où

   $W \geq \lceil B\, n / (B-A) \rceil$ (**15**)

   Où $\lceil x \rceil$ = le plus petit entier x égal ou supérieur à x.
2. Charge totale de la semaine : le nombre de salarié-jour par semaine doit être suffisant pour supporter la charge totale de la semaine. Si le salarié ne travaille que 5 jours par semaine :

   $5W \geq \sum_{j=1,7} n(j)$, donc $W \geq \lceil 1/5 * \sum_{j=1,7} n(j) \rceil$ (**16**)
3. Charge maximale journalière : le nombre de salariés disponible doit être suffisant pour respecter le pic des besoins une journée donnée. $W \geq \mathrm{Max}_j (n(j))$

Il faut une équipe dont la taille W égale le maximum des trois bornes inférieures.

b. Cas des multiples vacations (demandes jour de la semaine/week-end)

[Hun94] propose une méthode de calcul de la taille minimale d'une équipe. Chaque jour il y a P vacations (ex. P=3 pour le jour, soir et nuit) qui peuvent se chevaucher afin d'absorber un pic de besoins ou pour assurer une meilleure communication entre les postes.

Pendant la semaine, il faut au moins D(j) personnes sur la vacation j et au moins E(j) pour le week-end. En général, D(j) ≥ E(j) pour j=1,…,P. Chaque salarié ne travaille qu'une

seule vacation par jour et doit avoir au moins un repos avant de changer de vacation. Sur deux semaines consécutives, un salarié doit avoir 3 repos en une semaine et 4 repos sur l'autre semaine. Il doit aussi avoir au moins A week-ends de repos sur B, avec 0 <= A <= B. La durée maximale de travail ininterrompue est 5 jours. La méthode proposée ne distingue pas les différentes demandes suivant les jours de la semaine.

c. Cas des hiérarchies de compétences

Une situation fréquemment rencontrée est une hiérarchie de compétence. Au sommet de la hiérarchie, le niveau 1 est le plus qualifié : une telle personne peut remplacer toute autre personne du même niveau et également une personne aux niveaux inférieurs. Emmons et Burns sont les premiers à proposer [EB91] de calculer la taille de l'équipe dans ces conditions, tout en proposant un coût minimal. Les contraintes sont une seule vacation et les demandes sont confondues pour tous les jours de la semaine, y compris le week-end.

d. Cas des hiérarchies de compétences et multiples vacations

[Nar00] montre comment traiter le cas des multiples vacations et distingue les demandes des jours de la semaine et du week-end. Les variables sont :

Nombre de catégories = K,
Le nombre de salariés de la catégorie k = w(k)
Le besoin du week-end, pour la catégorie k et la vacation j = n(j, k)
Le besoin en semaine = d(j, k)
La taille cumulée des catégories entre 1 et k, $W(k) = \sum_{i=1,k} w(i)$
Le besoin cumulé des catégories entre 1 et k, $D(j, k) = \sum_{i=1,k} d(j , i)$

Par clarté d'écriture, on écrit $n(., k) = \sum_{j} n(j, k)$ et $d(., k) = \sum_{j} d(j, k)$

Méthode de Narasimhan [Nar00]

1. Borne inférieure basée sur la charge du week-end pour chaque catégorie : L1(k)

Le nombre de salariés de la catégorie k w(k) disponible le week-end doit satisfaire les besoins n(j, k). Pour que chaque salarié dispose de A week-end tous les B semaines, sur un cycle de B semaines, on a

$$(B - A)\, w(k) \geq B\, n(., k) \qquad (17)$$

Or, $w(k) \geq L1(k)$ par définition de la borne inférieure, d'où

$$L1(k) = \lceil B\, n(., k) / (B\text{-}A) \rceil \qquad (18)$$

2. Borne inférieure basée sur le besoin total de chaque catégorie : L2(k)

Le nombre total de salariés * jour par semaine disponible (quelle que soit la catégorie) doit être suffisant pour répondre aux besoins totaux de la semaine pour cette catégorie. Comme chaque salarié ne travaille que 4 jours par semaine,

$$4\, w(k) \geq 5\, d(., k) + 2\, n(., k) \qquad (19)$$

or, $w(k) \geq L2(k)$ par définition de la borne inférieure,
d'où $L2(k) = \lceil 1.25\ d(., k) + 0.5\, n(., k) \rceil$

La taille de la catégorie k égale au moins le maximum des deux bornes inférieures:

$$w(k) \geq \text{Max}\ \{ L1(k), L2(k) \} \qquad (20)$$

3. Borne inférieure basée sur le besoin cumulatif : L3(k)

Le nombre cumulatif des vacations disponibles des catégories 1 à k doit être au moins égal au nombre total de vacations demandées dans les catégories 1 à k

$$4\, W(k) \geq 5\, D(., k) + 2\, n(., k) \qquad (21)$$

d'où

$$W(k) \geq \lceil 1.25\, D(., k) + 0.5\, n(., k) \rceil \qquad (22)$$

Par ailleurs, le nombre cumulé de salariés des catégories 1 à k doit être au moins égal au nombre cumulé de 1 à k-1 plus le nombre w(k) :

$$W(k) = \text{Max}\ \{ W(k\text{-}1) + \text{Max}(L1(k), L2(k)) \} \qquad (23)$$

$L3(k) = \text{Max}\ \{ L3(k\text{-}1) + \text{Max}(L1(k), L2(k)), \lceil 1.25\, D(., k) + 0.5\, n(., k) \rceil \}$, si $k > 1$
$L3(1) = \text{Max}\ \{ L1(1), L2(1) \}$

e. Conclusion

Ces calculs sont basés sur des hypothèses rigides. En présence des contraintes supplémentaires, par exemple avec des contraintes de transition entre les vacations (pour des questions de repos) et les différents besoins par jour de la semaine, ils peuvent être utilisés pour proposer des bornes inférieures de la taille des équipes.

2.9.3 Le dimensionnement des vacations

L'art du dimensionnement peut être appliqué plus finement qu'au niveau de l'équipe : calculer le nombre de personnes dans chaque type de vacation (ou horaires types sur une journée). Cette possibilité a donné suite à toute une série de travaux sur les modèles implicites de construction de vacation, présentés au paragraphe § 2.1.

On constate que la complexité théorique du problème à résoudre est réduite de façon substantielle. Cela est dû au fait que les vacations ne sont pas décrites explicitement. Cependant, une fois ces nombres obtenus, il faut associer les heures de début et les pauses aux vacations et générer des plannings définitifs pour chaque salarié (Voir § 2.3 Construction de Tours Acycliques).

2.9.4 Le dimensionnement tout au long de la résolution

Enfin l'art du dimensionnement peut être appliqué tout au long du processus de la résolution, plus particulièrement dans le cas des algorithmes dits constructifs, où la solution est construite progressivement. Dans la résolution de problèmes de planification de grilles, cette approche a été proposée dans [CGL95].

Analyse de Caseau

Le problème consiste à affecter à chaque personne p à chaque jour w, à un code horaire a (ex. M= matin, S= soir, N= nuit, R= repos, J= jour), annoté par Affectation(p, w) = a. Les contraintes sont définies par les bornes Min / Max suivantes :

Par code horaire a par jour w, le total des affectations des agents doit être compris entre des bornes suivants (|Z| étant la cardinalité d'un ensemble Z) :

$$\text{Min}[w, a] \leq |\{ p \in \text{Personnes}, \text{Affectation}(p, w) = a\}| \leq \text{Max}[w, a]$$

Par code horaire a par personne p, le total des affectations doit être compris entre des bornes

$$\text{Min}[p, a] \leq |\{ w \in \text{Horizon}, \text{Affectation}(p, w) = a \}| \leq \text{Max}[p, a]$$

Soit N(a) = le nombre d'affectations au code a, sur l'ensemble des personnes et sur tout l'horizon. Par dimensionnement, N(a) doit être inférieur à la somme respectivement des maximum des affectations par personne et par jour au code a. De même il doit être supérieur à la somme respective des minima des affectations par personne et par jour. D'où :

$$\sum_{p \in \text{Personnes}} \text{Min}[p, a] \leq N(a) \leq \sum_{w \in \text{Horizon}} \text{Max}[w, a] \quad (\mathbf{24})$$

$$\sum_{w \in \text{Horizon}} \text{Min}[w, a] \leq N(a) \leq \sum_{p \in \text{Personnes}} \text{Max}[p, a] \quad (\mathbf{25})$$

Caseau propose les compteurs supplémentaires suivants :

Num[p, a] = no. d'affectations au code a pour la personne p sur l'horizon
Num[w, a] = no. d'affectations au code a pour le jour w sur l'ensemble des personnes
Pos[p, a] = no. d'affectations possibles au code a pour la personne p sur l'horizon
Pos[w, a]= no. d'affectations possibles au code a pour le jour w sur l'ensemble des personnes

[CGL95] montre que

Num[p, a] ≤ Max[p, a] Num[w, a] ≤ Max[w, a] le no. affecté doit être inférieur au max.
Min[p, a] ≤ Pos[p, a], Min[w, a] ≤ Pos[w, a], le no. possible doit être supérieur au min.

Caseau raffine la condition de dimensionnement par les équations suivantes :

$$\sum_{p \in \text{Personnes}} \max(\text{Num}[p,a], \text{Min}[p,a]) \leq N(a) \leq \sum_{w \in \text{Horizon}} \max(\text{Pos}[w,a], \text{Max}[w,a]) \quad (\mathbf{26})$$

$$\sum_{w \in \text{Horizon}} \max(\text{Num}[w,a], \text{Min}[w,a]) \leq N(a) \leq \sum_{p \in \text{Personnes}} \max(\text{Pos}[p,a], \text{Max}[p,a]) \quad (\mathbf{27})$$

Cette condition peut être utilisée pour élaguer les branches potentiellement infructueuses dans la construction de solution, à réévaluer chaque fois que Num ou Pos sont modifiés. Elle constitue une méthode de consistance globale.

Pour ce même problème, Caseau propose une condition supplémentaire applicable sur les compteurs Min et Max, liée au fait que le nombre total de jours ou de personnes est connu :

- Pour chaque code a et pour chaque personne,

$$\text{Max}[p,a] \leq |W| - \sum_{a' \neq a} \text{Min}[p,a'] \text{ et } \text{Min}[p,a] > |W| - \sum_{a' \neq a} \text{Max}[p,a'] \quad (\mathbf{28})$$

- Pour chaque code a et pour chaque jour,

$$\text{Max}[w,a] \leq |P| - \sum_{a' \neq a} \text{Min}[w,a'] \text{ et } \text{Min}[w,a] > |P| - \sum_{a' \neq a} \text{Max}[w,a'] \quad (\mathbf{29})$$

2.9.5 Conclusion sur le dimensionnement

Etape essentielle dans la constitution des équipes, le dimensionnement peut aussi être exploité pour détecter des insuffisances au cours de planification. Les propositions de Caseau permettent de réduire la complexité des problèmes combinatoires, et a fortiori des problèmes de planification des ressources humaines.

Nous retenons ces leçons dans le cadre de nos recherches au chapitre 5.