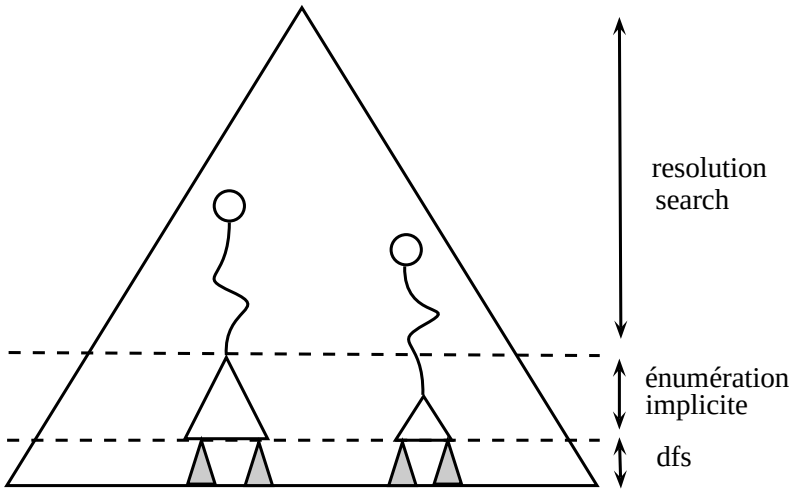
Coopération entre Resolution search et l'énumération implicite pour le sac à dos multidimensionnel en 0–1

Dans ce chapitre, nous présentons un méthode de résolution du MKP combinant Resolution search et une variante de l'algorithme d'énumération implicite présenté dans le chapitre 2. Cette méthode s'appuie également sur la prise en compte des coûts réduits à l'optimum de la relaxation continue et sur la décomposition de l'espace en hyperplans. La coopération est axée sur une exploration en trois niveaux de l'arbre de recherche : les branches hautes sont énumérées par Resolution search, les branches médianes par l'énumération implicite et les branches basses par l'algorithme de backtracking. Nous montrons que la structure de Resolution search permet d'explorer partiellement et de manière répétée les différents sous-problèmes associés aux hyperplans, ceci afin de diversifier la recherche. Nous proposons également une méthode alternative à la phase de remontée qui utilise des relations d'implication entre noeuds de l'arbre de recherche pour identifier des obstacles minimaux. Nous montrons que cette phase de remontée, baptisée *phase de remontée implicite*, accélère le processus de recherche dans le cas de notre implémentation fondée sur les coûts réduits. Les expériences numériques prouvent l'efficacité de cet algorithme : il a des performances comparables aux meilleures heuristiques connues en terme de calcul de minorant et résout des instances de taille moyenne en un temps plus rapide que les meilleures méthodes exactes connues sur ces mêmes instances. De plus, il résout les instances de grande taille à 10 contraintes et 500 variables de la `OR-Library`. Les valeurs optimales de ces instances étaient inconnues jusqu'à présent. Le travail présenté dans ce chapitre a fait l'objet d'une soumission à une revue internationale [9].

4.1 Principe général

D'un point de vue global, les branchements étant réalisés suivant l'ordre décroissant des coûts réduits, l'arbre de recherche est exploré suivant trois niveaux : Resolution search explore les branches hautes (variables à fort coût réduit), l'énumération implicite explore les branches médianes (variables à coût réduit moyen) et l'algorithme de backtracking explore les banches basses (variables de plus faible coût réduit et variables de base). Les deux méthodes coopèrent de la manière suivante : Resolution search fournit des sous–problèmes à l'énumération implicite qui, en les résolvant, permet d'obtenir des obstacles de petite taille utilisés par Resolution search pour guider l'exploration vers des zones prometteuses de l'espace de recherche.

Fig. 4.1 – Exploration de l'arbre de recherche en trois niveaux



4.2 Contributions apportées à Resolution search

Dans cette section, nous présentons deux modifications apportées à Resolution search. La première est une version à itérations limitées de l'algorithme qui permet d'explorer itérativement chaque sous–problème lié à la décomposition par hyperplans et la deuxième est une modification de la phase de remontée fondée sur des relations d'implication entre nœuds de l'arbre de recherche.

4.2.1 Exploration itérative de l'espace de recherche

Comme nous l'avons mentionné dans le chapitre 2, l'ordre dans lequel les hyperplans sont explorés influence fortement l'efficacité de la mise à jour de la borne inférieure. Nous allons voir que la structure de Resolution search présente un avantage intéressant qui permet de palier à ce problème.

Dans le chapitre 3, nous avons montré que la famille path–like $\mathscr{F}$ constitue un résumé des explorations passées : après chaque exécution de la boucle principale, $\mathscr{F}$ nous donne l'état de la recherche et le nœud suivant de l'exploration $u(\mathscr{F})$. Cette particularité permet de pouvoir interrompre le processus de recherche à un moment donné et de le récupérer ultérieurement sans perte d'information. De cette façon, il est possible de résoudre le problème P en effectuant de manière répétée un nombre d'itérations limité pour chaque sous–problème $P(k)$ jusqu'à ce que tous les sous–problèmes soient résolus. L'ordre d'exploration n'a alors plus d'importance car il n'est pas nécessaire qu'un sous–problème soit résolu pour commencer la résolution du sous–problème suivant. Les algorithmes 4.1 et 4.2 illustrent cette façon itérative de résoudre le problème.

Algorithme 4.1 – Algortithme Resolution search itératif

```
IRS(P,Nb_Iter, BI,F)
{
   iter = 0 ;
   while(iter < Nb_Iter) {
      try = obstacle(P, u(F),BI,S) ;
      if(try > BI)  BI = try ;
      ajouter S à F et mettre à jour F ;
      if((*,*,...,*) ∈ F)   Break ;
      iter++;
   }
}
```

Algorithme 4.2 – Exploration des hyperplans

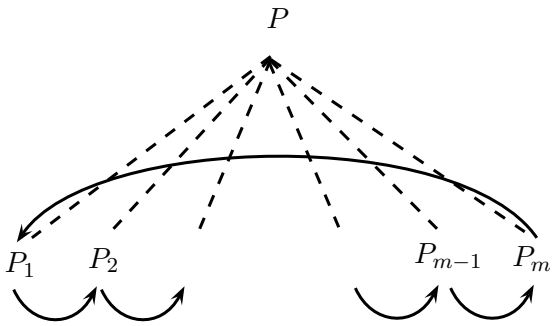
```
resoudre_MKP() {
   BI = glouton() ;
   Calculer les bornes k_min et k_max ;
   Définir K = {k_min , ... , k_max} ;
   for(k = k_min, ..., k_max) CFamily[k] = ∅
   While (K ≠ ∅) {
      Choisir k ∈ K ;
      Choisir Nb_Iter_k ≥ 1 ;
      IRS(P(k),Nb_iter_k,BI,CFamily[k]) ;
      if((*,*,...,*) ∈ CFamily[k])   K = K − {k} ;
   }
}
```

La fonction `IRS` de l'algorithme 4.1 prend en arguments : un problème P, un nombre d'itérations limité `Nb_Iter`, un minorant `BI` et une famille path–like $\mathscr{F}$. Cet algorithme consiste à effectuer un nombre `Nb_Iter` d'itérations de Resolution search.

La fonction `resoudre_MKP()` de l'algorithme 4.2 consiste à résoudre le problème MKP en utilisant cette version iterative de Resolution search. L'algorithme calcule une borne inférieure gloutonne afin de déduire les bornes k_{min} et k_{max}, puis effectue un certain nombre d'itérations de `IRS` sur chaque sous–problème $P(k)$. La variable CFamily[k] correspond à la famille path–like de `IRS` pour le problème $P(k)$. Lorsque l'un des problèmes $P(k)$ est résolu, c'est–à–dire que $(*, *, ..., *) \in$ CFamily[k], il est éliminé de la recherche. Le problème P est alors résolu lorsque tous les sous–problèmes $P(k)$ sont éliminés.

Ce type d'exploration peut être étendu à une quelconque décomposition $P_1, P_2, ..., P_m$ du problème P original. Il peut par exemple être utilisé pour la résolution itérative de sous–problèmes pour lesquels certaines variables sont fixées. La figure 4.2 illustre cette approche de résolution.

Fig. 4.2 – Exploration itérative des sous–problèmes



4.2.2 Phase de remontée implicite

Dans le papier original de Resolution search, Chvátal propose de libérer une à une les variables précédemment instanciées dans la phase de descente (sauf la dernière) et de vérifier à chaque fois la faisabilité de la solution partielle correspondante en exécutant la fonction `oracle` (cf. chapitre 3). Cette phase de remontée correspond à la portion de code suivante de l'algorithme 3.4 :

```
while(la pile n'est pas vide){
   dépiler j ;
   S_j = * ;
   if(oracle(S) > BI) S_j = u_j^+ ;
}
```

Nous avons montré par des expérimentations que cette portion de l'algorithme ralentit considérablement le processus de recherche (cf. chapitre 3 section 3.9.1). Nous proposons un autre moyen d'identifier la clause S que nous appelons *phase de remontée implicite*.

Contrairement à la phase de remontée originale, cette méthode ne cherche pas à libérer des variables une fois l'échec rencontré, mais identifie, lors de la phase de descente, un ensemble d'instanciations qui ne seront pas contenues dans le futur obstacle. Cette méthode est basée sur des relations d'implication existantes entre certains nœuds de l'arbre de recherche.

Supposons que l'instanciation des variables dans le nœud u implique la fixation de variables supplémentaires constituant le nœud v. En d'autres termes, supposons que $u \Rightarrow v$. Si, pendant la phase de descente, u est étendu en $u^+ = u \cup v \cup w$ et que u^+ viole une ou plusieurs contraintes du problème, la clause S des affectations responsables de l'échec est réduite à $u \cup w$.

Proposition 4.1 *Soit u, v et w des vecteurs de $\{0,1,*\}^n$. Si $u \Rightarrow v$ et si $u \cup v \cup w$ est un obstacle, alors $u \cup w$ est un obstacle.*

Preuve Nous allons prouver la proposition par induction sur la longueur de v et par l'absurde. Supposons que v soit de longueur 1 tel que $v = \{x_j\}$, donc $u' = u \cup \{\bar{x}_j\}$ est un obstacle puisque $u \Rightarrow v$ et $u'' = u \cup \{x_j\} \cup w$ est un obstacle par hypothèse. Alors, la résolvante de u' et u'' qui est $u' \nabla u'' = u \cup w$ est un obstacle. Supposons à présent que la proposition soit vraie pour un vecteur v de longueur k, nous allons prouver que la proposition est toujours vraie si v est de longueur $k+1$. Soit $v = v' \cup \{x_j\}$, alors $u' = u \cup v' \cup \{\bar{x}_j\}$ est un obstacle et $u'' = u \cup v' \cup \{\bar{x}_j\} \cup w$ est également un obstacle. Donc la résolvante de u' et u'' qui est $u' \nabla u'' = u \cup v' \cup w$ est un obstacle et par hypothèse d'induction, $u \cup w$ est un obstacle. □

Bien que la phase de remontée implicite puisse remplacer la phase de remontée originale de Resolution search, son efficacité dépend du problème résolu et des relations d'implication qui peuvent être déduites entre les nœuds de l'arbre de recherche.

Dans le cas de la résolution du MKP, nous proposons une implémentation fondée sur les relations d'implication qui peuvent être déduites de la contrainte des coûts réduits. Dans le cas où le nœud de départ $u(\mathscr{F})$ donné à la fonction `obstacle` contient des instanciations des variables à l'opposé de leur valeur optimale, *gap* est diminuée de la valeur des coûts réduits correspondants et cette diminution implique la fixation des variables non instanciées dans $u(\mathscr{F})$ ayant un coût réduit supérieur à *gap*. D'après la proposition 4.1, les variables ainsi fixées à cause de la diminution de *gap* engendrée par $u(\mathscr{F})$ ne sont alors pas prises en compte dans la clause S. L'algorithme 4.3 détaille l'application de la phase de remontée implicite dans la fonction `obstacle`.

Algorithme 4.3 – Fonction obstacle avec phase de remontée implicite pour le MKP

```
obstacle(P,u,BI,S)
{
   gap = v(P̄)− BI ;
   for(j = 1,2,...,n)
      if(u_j = 1 − x̄_j) gap = gap − c̄_j ;
   u+ = u ;S = u+ ;
   borne = oracle(u+) ;
   if(borne > BI){
      //Phase de remontée implicite
      for(j = 1,2,...,n)
         if(u_j = ∗ et c̄_j > gap) u+_j = x̄_j ;
      //Phase de descente
      while(u+ ∉ {0,1}^n) {
         choisir un indice j tel que u+_j = ∗ et une valeur v dans {0,1} ;
         u+_j = v ;S_j = v ;
         borne = oracle(u+) ;
         if(borne ≤ BI) break ;
      }
      if(borne > BI) BI = borne ;
   }
   return borne ;
}
```

Nous avons mené une expérimentation de Resolution search avec et sans phase de remontée implicite afin d'en évaluer l'efficacité. Le tableau 4.1 donne les résultats obtenus sur les instances cb5.100 de la `OR-Library`. La colonne *opt.* donne le nombre moyen d'appels à `oracle` nécessaires à l'obtention de la valeur optimale et la colonne *preuve* donne le nombre moyen d'appels à `oracle` nécessaires à la preuve d'optimalité.

Tab. 4.1 – Impact de la phase de remontée implicite

Instance	sans remontée implicite		avec remontée implicite	
	opt.	preuve	opt.	preuve
0-9	69286	145395	**55726**	**111865**
10-19	71286	155152	**57736**	**122944**
20-29	17414	45696	**13891**	**39376**

On constate une nette diminution du nombre moyen d'appels à `oracle` pour l'obtention de la valeur optimale mais également pour la preuve d'optimalité. Si l'on compare la somme des appels à `oracle` pour l'ensemble des 30 instances cb5.100, on constate une diminution de l'ordre de $19,3\%$ pour l'obtention de la valeur optimale et une diminution de l'ordre de $20,8\%$ pour l'obtention de la preuve d'optimalité.

4.3 Description du processus d'exploration

Dans cette section, nous décrivons le principe d'exploration de la fonction `obstacle`. Nous détaillons l'application de la phase de remontée implicite ainsi que l'hybridation avec l'algorithme d'énumération implicite.

Partant du nœud $u(\mathscr{F})$ fourni par la famille path–like $\mathscr{F}$, `obstacle` remplace pas–à–pas les composantes $*$ de $u(\mathscr{F})$ par 0 ou 1 de manière à construire le nœud u^+. Si u^+ devient un obstacle (alors noté u^*), la procédure cherche une clause minimale S telle que $S \sqsubseteq u^*$ et S est un obstacle. Afin d'illustrer le déroulement de la fonction `obstacle`, nous considérons le problème P^d suivant :

$$
\begin{array}{lrcrcrcrcrcl}
(P^d)\ \text{Maximiser} & 10x_1 & + & 10x_2 & + & 8x_3 & + & 8x_4 & + & 7x_5 & & \\
\text{sujet à} & 5x_1 & + & 2x_2 & + & 10x_3 & + & 1x_4 & + & 3x_5 & \leq & 10 \\
& 2x_1 & + & 11x_2 & + & 2x_3 & + & 10x_4 & + & x_5 & \leq & 11 \\
& x = {}^T(x_1, x_2, x_3, x_4, x_5) \in \{0,1\}^5 & & & & & & & & & &
\end{array}
$$

Supposons que l'on cherche à résoudre le problème $P^d(2)$ qui correspond au problème P^d avec la contrainte additionnelle $(1 \cdot x = 2)$. La résolution de la relaxation linéaire de $P^d(2)$ nous donne une solution optimale $\bar{x}$ de valeur $\texttt{BS} = 19.55$ et un vecteur de coûts réduits $\bar{c}$ tels que :

$$\bar{x} = (1, 0.78, 0.22, 0, 0) \qquad \text{et} \qquad \bar{c} = (2, 0, 0, -1.78, -0.78)$$

Supposons que nous connaissions un minorant $\texttt{BI} = 16$ du problème, alors la contrainte des coûts réduits s'écrit

$$2(1 - x_1) + 1.78x_4 + 0.78x_5 \leq 2$$

Au début de la procédure, la relaxation linéaire du problème est résolue pour chaque hyperplan disponible dans le but de fournir l'information nécessaire à la génération de la contrainte des coûts réduits.

La première étape, appelée *Phase de vérification*, consiste à vérifier la faisabilité de $u(\mathscr{F})$. Si une contrainte est violée, la vérification s'arrête et l'affectation partielle correspondante, représentant un obstacle, est mémorisée dans $\mathscr{F}$ sous forme de clause. Simultanément, la contrainte des coûts réduits est vérifiée et la valeur *gap* est mise à jour : pour chaque variable hors base fixée à l'opposé de sa valeur optimale $(1 - \bar{x}_j)$, son coût réduit $(\bar{c}_j)$ est soustrait à *gap* et si $gap < 0$, l'affectation partielle courante est un obstacle. Dans ce cas, S est uniquement composée des instanciations des variables fixées à l'opposé de leur valeur optimale dans $u(\mathscr{F})$. Par exemple, supposons que lors de la résolution de $P^d(2)$, $u(\mathscr{F}) = (0, 0, 0, 1, *)$, nous avons gap $= -1.78$ donc $u(\mathscr{F})$ est un obstacle. La solution partielle $(0, *, *, 1, *) \sqsubseteq (0, 0, 0, 1, *)$ qui viole la contrainte des coûts réduits est également un obstacle et dans ce cas $S = x_1\bar{x}_4$ est ajoutée à $\mathscr{F}$.

Nous passons ensuite à la deuxième étape qui constitue l'application de la phase de remontée implicite décrite en section 4.2.2. Cette phase consiste à fixer toutes les variables libres hors base de coût réduit strictement supérieur à *gap* à leur valeur optimale. Ces variables doivent en effet être fixées à leur valeur optimale pour satisfaire la contrainte des coûts réduits. Fixer ces variables à leur valeur optimale n'est donc qu'une conséquence des instanciations des variables dans $u(\mathscr{F})$. Selon la proposition 4.1, elles peuvent ne pas être incluses dans S. Pour illustrer ce processus, supposons que `obstacle` commence avec $u(\mathscr{F}) = (0, *, *, *, *)$, x_1 étant fixé à $(1 - \bar{x}_1)$, on a $gap = 0$. Puisque $\bar{c}_4 = 1.78$ est supérieur à *gap*, x_4 doit être fixée à $\bar{x}_4 = 0$. Nous avons donc $u(\mathscr{F}) = (0, *, *, *, *)$ implique $u^+ = (0, *, *, 0, *)$ et $x_4 = 0$ ne sera pas inclue dans S.

La phase suivante constitue la phase de descente (ou *waxing phase*) telle que décrite dans le papier original [14]. Cette phase consiste à assigner des valeurs aux variables libres jusqu'à atteindre un obstacle. La stratégie de branchement choisie consiste à sélectionner la variable libre de plus grande valeur absolue de coût réduit et à lui assigner sa valeur optimale $\bar{x}_j$. Chaque fois qu'un branchement est réalisé, la réalisabilité de l'affectation partielle courante est vérifiée et en cas d'echec, la phase de descente s'arrête et la clause S correspondante est ajoutée à $\mathscr{F}$. Si nous considérons l'exemple précédent, $u^+ = (0, *, *, 0, *)$ après la phase de remontée implicite. Puisque x_5 est la variable libre de plus grande valeur absolue de coût réduit ($|\bar{c}_5| = 0,78$), nous la fixons à sa valeur optimale ($u_5^+ = 0$).

Lorsque le nombre de variables libres restant est inférieur ou égal à un paramètre Δ donné, la phase de descente s'arrête et le sous–problème correspondant est résolu par l'énumération implicite. Ce sous–problème inclut les variables libres de faible coût réduit et les variables de base. L'énumération implicite explorant entièrement le sous–arbre correspondant aux variables libres, la clause S générée ne contient aucun des branchements réalisés pendant cette phase. Seules les branchement réalisés pendant la phase de vérification et / ou la phase de descente sont considérées pour la construction de la clause S. Dans l'exemple ci–dessus, l'énumération des variables restantes x_2 et x_3 donne les configurations suivantes :

$u^* = (0, 0, 0, 0, 0)$, $c \cdot u^* \leq \texttt{BI}$,

$u^* = (0, 0, 1, 0, 0)$, $c \cdot u^* \leq \texttt{BI}$,

$u^* = (0, 1, 0, 0, 0)$, $c \cdot u^* \leq \texttt{BI}$,

$u^* = (0, 1, 1, 0, 0)$, $A \cdot u^* > b$: irréalisable.

Puisqu'aucune de ces configurations n'améliore la meilleure solution connue, la clause $S = \bar{x}_1 \bar{x}_5$ est ajoutée à $\mathscr{F}$. Notons que Δ peut être mise à jour dynamiquement durant la recherche, selon que l'on veut favoriser la partie Resolution search ou la partie énumération implicite.

L'algorithme utilisé pour énumérer les variables du sous–problème constitué des variables de faible coût réduit et des variables de base est similaire à l'énumération implicite présentée dans le chapitre 2.

L'unique modification apportée a été de retirer à cet algorithme la propagation des coûts réduits effectuée à chaque nœud (cf. chapitre 2, section 2.4.3). En effet, dans le cas de la résolution de ces sous–problèmes constitués essentiellement de variables à faible coût réduit, l'expérimentation nous a montré que cette propagation perd de son efficacité et a tendance à ralentir le processus de recherche. Comme mentionné précédemment, cet algorithme oriente la recherche vers les solutions les plus à même de violer la contrainte des coûts réduits. La politique de branchements à chaque nœud u peut se résumer de la manière suivante :

1. Résoudre le problème $\bar{P}(k, u)$.
2. Brancher sur la variable libre hors base (x_j) de plus grande valeur absolue de coût réduit ($|\bar{c}_j|$) en la fixant à l'opposé de sa valeur optimale ($1 - \bar{x}_j$).
3. Mettre à jour $gap = \lfloor \texttt{BS}(u) \rfloor - \texttt{BI} - 1$ par $gap = gap - \bar{c}_j$.
4. Fixer toutes les variables hors base de coût réduit strictement supérieur à gap à leur valeur optimale.

A chaque nœud, la règle de branchement cherche à provoquer la violation de la contrainte des coûts réduits de manière à élaguer au plus tôt l'arbre de recherche. Nous avons montré dans le chapitre 2 que cette méthode est efficace uniquement lorsqu'un bon minorant du problème est connu. Dans le cas de l'hybridation avec Resolution search, elle correspond bien à la stratégie générale : utiliser Resolution search pour explorer l'espace de recherche de manière pertinente et l'énumération implicite pour résoudre efficacement des sous–problèmes de petite taille.

4.4 Expériences numériques

Les expériences numériques ont été réalisées sur les instances du jeu Chu et Beasley de la `OR-Library`. Les résultats sont comparés avec ceux obtenus par l'énumération implicite du chapitre 2 ainsi qu'avec le solveur commercial CPLEX 9.2. Le paramètre Δ est incrémenté toutes les 100 itérations partant de 20 jusqu'à 60, et réinitialisé à 20 si la borne inférieure est améliorée. Les résultats obtenus sur les instances cb10.250, cb5.500 et cb10.500 sont présentés dans les tableaux 4.2, 4.3 et 4.4. La description des données par colonne est la suivante :

- *Instance* est le nom de l'instance.
- z^{opt} est la valeur optimale.
- *opt.* est le temps de calcul requis en secondes pour obtenir une solution optimale.
- *preuve* est le temps de calcul requis en secondes pour prouver l'optimalité de la meilleure solution trouvée.

La colonne *RSBB* correspond aux résultats obtenus avec l'hybridation de Resolution search et de l'énumération implicite, la colonne *enum [69]* correspond aux résultats obtenus avec l'énumération implicite présentée dans le chapitre 2 et la colonne *CPLEX* correspond

aux résultats obtenus par CPLEX 9.2 (ERROR si CPLEX a dépassé la capacité de mémoire). Les résultats obtenus sur les instances à 10 contraintes et 250 variables sont exposés dans le tableau 4.2 et ceux obtenus sur les instances à 5 contraintes et 500 variables sont exposés dans le tableau 4.3. Dans le tableau 4.4, correspondant aux instances à 10 contraintes et 500 variables, la colonne $z^{opt} - \underline{z}$ donne la différence entre la valeur optimale trouvée par notre algorithme et le meilleur minorant connu jusqu'à présent : (vv) indique qu'il y a eu une amélioration par rapport aux résultats de Vasquez et Vimont [68] et (wh) indique qu'il y a eu une amélioration par rapport aux résultats de Wilbaut et Hanafi [72]. Cette méthode résout l'ensemble des instances cb10.250 et cb5.500 plus rapidement que l'énumération implicite présentée dans le chapitre 2 et que CPLEX 9.2. De plus, elle résout l'ensemble des instances cb10.500 dont les valeurs optimales étaient inconnues jusqu'à présent.

Tab. 4.2 – Résultats obtenus par l'algorithme hybride sur les instances cb10.250

Instance	z^{opt}	RSBB		enum [69]	CPLEX
		opt.	preuve	preuve	preuve
cb10.250_0	59187	1	**3326**	5126	ERROR
cb10.250_1	58781	325	**1510**	43910	4369
cb10.250_2	58097	2	**795**	1509	6746
cb10.250_3	61000	1906	**6430**	9080	ERROR
cb10.250_4	58092	738	**20749**	28789	ERROR
cb10.250_5	58824	6	**747**	1099	7394
cb10.250_6	58704	0	**647**	1052	8255
cb10.250_7	58936	3426	**58294**	87329	ERROR
cb10.250_8	59387	335	**2391**	4428	ERROR
cb10.250_9	59208	0	**4446**	6902	ERROR
cb10.250_10	110913	560	**3593**	4952	ERROR
cb10.250_11	108717	630	**3521**	7465	ERROR
cb10.250_12	108932	4	**2141**	3382	ERROR
cb10.250_13	110086	91	**9573**	15091	ERROR
cb10.250_14	108485	0	**1585**	2306	9869
cb10.250_15	110845	94	**4130**	5954	ERROR
cb10.250_16	106077	34	**5130**	7867	ERROR
cb10.250_17	106686	816	**3362**	5538	ERROR
cb10.250_18	109829	17	**2930**	4817	ERROR
cb10.250_19	106723	24	**807**	1384	5716
cb10.250_20	151809	353	**584**	973	4695
cb10.250_21	148772	0	**1559**	2284	ERROR
cb10.250_22	151909	0	**292**	828	5048
cb10.250_23	151324	318	**563**	5489	2234
cb10.250_24	151966	533	**812**	932	5727
cb10.250_25	152109	3	**323**	818	2826
cb10.250_26	153131	7	**50**	239	374
cb10.250_27	153578	0	**3279**	8469	ERROR
cb10.250_28	149160	109	**363**	1276	1638
cb10.250_29	149704	0	**413**	826	2487

TAB. 4.3 – Résultats obtenus par l'algorithme hybride sur les instances cb5.500

Instance	z^{opt}	RSBB		enum [69]	CPLEX
		opt.	preuve	preuve	preuve
cb5.500_0	120148	60	**109**	1331	8001
cb5.500_1	117879	9	**15**	1034	855
cb5.500_2	121131	5	**35**	1112	4687
cb5.500_3	120804	14	**47**	630	6050
cb5.500_4	122319	5	**27**	558	1578
cb5.500_5	122024	45	**67**	932	3678
cb5.500_6	119127	4	**51**	851	6937
cb5.500_7	120568	7	**27**	804	2672
cb5.500_8	121586	48	**77**	1172	ERROR
cb5.500_9	120717	1	**44**	2783	5756
cb5.500_10	218428	53	**63**	838	1457
cb5.500_11	221202	1	**11**	2146	905
cb5.500_12	217542	221	**232**	1634	3728
cb5.500_13	223560	1	**82**	1211	5009
cb5.500_14	218966	8	**11**	486	485
cb5.500_15	220530	21	**40**	814	3122
cb5.500_16	219989	10	**22**	593	1211
cb5.500_17	218215	1	**18**	811	1096
cb5.500_18	216976	1	**37**	697	2373
cb5.500_19	219719	6	**67**	969	2522
cb5.500_20	295828	1	**5**	767	47
cb5.500_21	308086	20	**44**	674	475
cb5.500_22	299796	1	**10**	783	187
cb5.500_23	306480	8	**22**	620	1182
cb5.500_24	300342	1	**27**	610	204
cb5.500_25	302571	10	**16**	634	988
cb5.500_26	301339	5	**6**	466	366
cb5.500_27	306454	3	**6**	558	148
cb5.500_28	302828	3	**18**	724	367
cb5.500_29	299910	59	**93**	560	478

Les résultats obtenus illustrent les points positifs suivants de notre approche : (i) en dépit des temps de calculs parfois très élevés, notre algorithme n'a jamais dépassé la capacité de mémoire et (ii) les temps de calcul requis pour trouver les valeurs optimales son relativement faibles. En effet, nous avons comparé les minorant obtenus par notre approche avec ceux obtenus par l'heuristique de Wilbaut et Hanafi [72] en un temps limité à 2 heures[1]. On considère qu'une méthode est plus efficace qu'une autre si un même minorant a été trouvé en un temps plus rapide ou si un meilleur minorant a été trouvé dans le temps imparti.

[1]D'autres heuristiques performantes ont également été proposées par Vasquez et Hao [67] et Vasquez et Vimont [68] cependant les temps d'obtention des meilleurs minorants sont quelquefois long (entre 300 secondes et 25 heures) alors que les résultats publiés par Wilbaut et Hanafi sont ceux obtenus en 2 heures de temps de calcul. C'est pourquoi nous comparons nos résultats avec cette approche

Tab. 4.4 – Résultats obtenus par l'algorithme hybride sur les instances cb10.500

Instance	z^{opt}	opt.	preuve	$z^{opt} - \underline{z}$
cb10.500_0	117821	5525	2041200	+**10**(vv)
cb10.500_1	119249	246521	982776	+**17**(vv)
cb10.500_2	119215	4200	2764800	0
cb10.500_3	118829	170695	322744	+**4**l(wh)
cb10.500_4	116530	19394	9108000	+**16**(wh)
cb10.500_5	119504	8375	676837	0
cb10.500_6	119827	9892	461009	0
cb10.500_7	118344	582278	646591	+**11**(wh)
cb10.500_8	117815	310868	791701	0
cb10.500_9	119251	11423	1277833	0
cb10.500_10	217377	18	1856970	0
cb10.500_11	219077	2083	1575519	0
cb10.500_12	217847	18	20129	0
cb10.500_13	216868	67	376181	0
cb10.500_14	213873	7345	4975200	+**14**(vv)
cb10.500_15	215086	123	158186	0
cb10.500_16	217940	48419	130088	0
cb10.500_17	219990	543060	1255820	0
cb10.500_18	214382	45998	208335	+**7**(vv)
cb10.500_19	220899	743	76686	0
cb10.500_20	304387	23938	29756	0
cb10.500_21	302379	58	30392	0
cb10.500_22	302417	241986	380031	+**1**(vv)
cb10.500_23	300784	3345	13682	0
cb10.500_24	304374	683	60619	0
cb10.500_25	301836	107128	111286	0
cb10.500_26	304952	169	66919	0
cb10.500_27	296478	4122	33730	0
cb10.500_28	301359	29488	140984	0
cb10.500_29	307089	4324	16085	0

Notre méthode est plus efficace que l'heuristique de Wilbaut et Hanafi pour l'ensemble des instances cb5.500 et elle est plus efficace pour 16 des 30 instances cb10.500.

4.5 Amélioration du calcul de minorants

Nous avons expérimenté une méthode consistant à renforcer la contrainte des coûts réduits sur chaque hyperplan afin d'améliorer le calcul de minorants. Pour ce faire, nous proposons de réduire le membre de droite de la contrainte des coûts réduits $gap = \lfloor \mathtt{BS} \rfloor - \mathtt{BI} - 1$ d'une valeur ρ puis de diminuer ρ successivement après un certain nombre d'itérations jusqu'à obtenir le membre de droite initial. L'intérêt de cette démarche est que la diminution du membre de droite de la contrainte des coûts réduits favorise la fixation de variables et l'élagage de l'arbre de recherche.

Notons qu'à chaque fois que la valeur ρ est modifiée, les familles path–like de chaque hyperplan sont vidées de leur clauses. Cependant, afin d'éviter la redondance de la recherche, certaines clauses sont conservées : pour les 3 hyperplans de plus grande valeur $v(\bar{P}(k))$, nous conservons les clauses C_i pour lesquelles $\texttt{oracle}(C_i) \leq \texttt{BI}$. Expérimentalement, nous avons testé cette méthode sur les instances cb10.500 de la `OR-Library`. Nous avons conclu des premières expérimentations qu'initialiser l'écart $\rho = 500$ puis le diminuer de 50 toutes les 200 secondes donnait des résultats intéressants. Il est probable qu'un autre paramétrage donne de meilleurs résultats mais cette expérimentation a simplement pour but de donner une première évaluation de cette méthode.

Le tableau 4.5 montre les résultats obtenus en comparaison avec l'heuristique de Wilbaut et Hanafi [72]. Dans ce tableau 4.5, la colonne *BI* correspond au meilleur minorant trouvé et *t(s)* est le temps de calcul requis pour trouver BI. Les colonnes *Wilbaut, Hanafi [72]* correspondent aux résultats trouvés par l'heuristique de Wilbaut et Hanafi, *RSBB* correspond aux résultats obtenus par la méthode hybride présentée dans ce chapitre en limitant son temps d'exécution à 2 heures et *RSBBC* correspond aux résultats obtenus en faisant varier le membre de droite de la contrainte des coûts réduits. Les valeurs en gras indiquent une amélioration soit en temps (un même minorant a été trouvé en un temps plus rapide) soit en valeur (un meilleur minorant a été trouvé dans le temps imparti).

Ces résultats montrent les possibilités d'amélioration de cette méthode hybride pour la calcul de minorants. Avec cette première méthode, nous avons en effet obtenu une amélioration de 25 instances sur les 30 instances cb10.500 par rapport à l'heuristique de Wilbaut et Hanafi.

4.6 Conclusion

Nous avons proposé, dans ce chapitre, certaines contributions à Resolution search. La première contribution concerne une modification de la phase de remontée originale, fondée sur des relations d'implication qui peuvent être déduites entre certains nœuds de l'arbre de recherche. Cette phase de remontée implicite, permet de procéder à l'identification d'obstacles minimaux à moindre coût par rapport à la phase de remontée originale. Nous avons vu expérimentalement qu'elle offre une réduction sensible du nombre d'appels à `oracle` lors de la résolution d'instances de MKP. Nous avons également montré que la structure de Resolution search permet d'améliorer la diversification de la recherche en explorant partiellement et de manière répétée plusieurs sous–problèmes du problème original. Nous avons proposé une méthode hybride qui combine Resolution search à un algorithme d'énumération implicite inspiré des travaux du chapitre 2. Cette hybridation constitue une réelle coopération entre Resolution search, qui guide la recherche vers des zones prometteuses de l'espace, et l'énumération implicite qui, en résolvant efficacement des sous–problèmes de petite taille, fournit des obstacles intéressants à Resolution search. Les expérimentations ont montré que la méthode résultante est capable de résoudre les instances cb10.250 et cb5.500

en un temps plus rapide que les meilleures méthodes exactes connues sur ces instances et d'obtenir les preuves d'optimalité des instances cb10.500 qui étaient inconnues jusqu'à présent. Nous avons vu qu'il était possible, en resserrant la contrainte des coûts réduits, d'améliorer le calcul de minorants de cette méthode. Des expérimentations sur les instances cb10.500 ont montré qu'elle donnait de meilleurs résultats que l'une des heuristiques les plus performantes connues. Bien que l'algorithme proposé dans ce chapitre soit relativement éloigné de l'implémentation originale de Resolution search proposée par Chvátal, il montre que cette méthode constitue un schéma d'exploration intéressant pour la résolution de problèmes d'optimisation linéaire à variables binaires.

Tab. 4.5 – Amélioration du calcul de minorants pour les instances cb10.500

Instance	Wilbaut, Hanafi [72]		RSBB		RSBBC	
	BI	t (s)	BI	t (s)	BI	t (s)
cb10.500_0	117809	3086	117809	141	**117809**	**13**
cb10.500_1	119217	2068	119222	5180	**119222**	**4059**
cb10.500_2	**119215**	**5028**	119211	410	119211	11
cb10.500_3	118801	3400	118813	860	**118813**	**26**
cb10.500_4	116514	241	116514	544	**116514**	**36**
cb10.500_5	119490	3130	119504	8375	**119504**	**4556**
cb10.500_6	**119794**	**2810**	119827	9862	119790	103
cb10.500_7	**118333**	**6146**	118323	1	118323	3
cb10.500_8	**117781**	**2200**	117779	10	117779	8
cb10.500_9	119251	4282	119251	11423	**119251**	**147**
cb10.500_10	217377	9	217377	18	**217377**	**4**
cb10.500_11	219077	1727	219077	2083	**219077**	**123**
cb10.500_12	217847	428	217847	18	**217847**	**7**
cb10.500_13	216868	1032	216868	67	**216868**	**7**
cb10.500_14	213853	1707	213861	1127	**213861**	**1067**
cb10.500_15	215086	1317	215086	123	**215086**	**14**
cb10.500_16	217940	4606	217915	1973	**217940**	**2944**
cb10.500_17	**219990**	**5729**	219984	73	219984	21
cb10.500_18	214375	329	214375	196	**214382**	**6014**
cb10.500_19	220886	3174	220899	744	**220899**	**659**
cb10.500_20	304387	5118	304363	40	**304387**	**4041**
cb10.500_21	302379	1462	302379	58	**302379**	**17**
cb10.500_22	302416	85	302416	12	**302416**	**22**
cb10.500_23	300784	2004	300784	3345	**300784**	**40**
cb10.500_24	304374	1561	304374	683	**304374**	**64**
cb10.500_25	301836	5314	301796	13	**301836**	**2648**
cb10.500_26	304952	326	304952	169	**304952**	**26**
cb10.500_27	296472	785	296478	4122	**296478**	**1168**
cb10.500_28	301357	1015	301353	3141	**301359**	**1867**
cb10.500_29	307089	839	307089	4324	**307089**	**17**

Chapitre 5

Application de Resolution search au problème de planification de techniciens et d'interventions pour les télécommunications

Nous présentons, dans ce chapitre, un travail réalisé dans le cadre du $5^{\text{ème}}$ challenge de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF)[1]. Le sujet, proposé par *France Télécom*, concernait la planification de techniciens et d'interventions dans le domaine des télécommunications. Nous détaillons, dans une première partie, l'algorithme de recherche de bornes supérieures que nous avons présenté pour ce challenge et nous exposons ensuite une étude réalisée a posteriori sur l'application de Resolution search à la résolution d'un sous–problème du problème général.

L'objectif du problème que l'on note TIST (pour *Technicians and Interventions Scheduling Problem for Telecommunications*) est de concevoir des ordonnancements de techniciens et d'interventions dans le but d'aider les décideurs de *France Télécom* dans la réalisation de leurs plannings. Chaque jour, des équipes de techniciens doivent être formées afin d'effectuer différentes tâches à différents lieux géographiques. L'accroissement du nombre de clients et la diversification des services dus au développement du haut débit comme la voix sur ip ou la télévision par Internet rendent la réalisation des plannings de plus en plus complexe.

Les interventions sont caractérisées par des critères spécifiques comme une priorité de planification et une durée d'exécution, et certaines interventions doivent être réalisées avant d'autres. Les interventions sont également composées de différentes tâches qui nécessitent un certain nombre de techniciens d'un certain niveau de compétence dans un domaine donné. Les techniciens sont spécialisés dans différents domaines avec différents niveaux de compétence et chaque technicien a une liste de jours d'indisponibilité. D'autre part, chaque

[1] `http ://www.g-scop.fr/ChallengeROADEF2007/` ou `http ://www.roadef.org/`

intervention a un coût donné si elle est sous–traitée par une entreprise extérieure. Le coût total de ces interventions sous–traitées ne peut dépasser un certain budget. Le TIST consiste à ordonnancer un ensemble d'interventions en minimisant une fonction scalaire qui attribue une pénalité plus grande aux interventions les plus prioritaires. Ce problème a deux aspects combinatoires distincts : l'ordonnancement des interventions (qui dépend des contraintes de précédence) et la construction des équipes de techniciens (qui dépend du jour courant).

Dans une première partie, nous décrivons l'algorithme de résolution que nous avons présenté lors du challenge ROADEF'07. Cet algorithme, basé sur la métaheuristique GRASP (*Greedy Randomized Adaptive Search Procedure* [**?**]), donne des résultats prometteurs et nous a permis d'être classé $4^{ème}$ sur 35 équipes participantes lors du classement final. Néanmoins, l'un des défauts majeurs de cet algorithme concerne le choix des interventions à sous–traiter. Ce choix est effectué par une heuristique simple consistant à choisir dès le départ un ensemble d'interventions à sous–traiter et à ne plus considérer ces interventions durant la recherche. Un mauvais choix de départ peut donc pénaliser l'efficacité de l'algorithme durant tout le processus de recherche. Nous avons réalisé une étude a posteriori sur ce problème dans le but d'améliorer les résultats précédemment obtenus. Nous présentons, dans la deuxième partie de ce chapitre, les différentes voies qui ont été explorées pour cette étude et montrons en particulier que l'utilisation de Resolution search permet d'obtenir de meilleurs résultats que les autres méthodes testées sur la résolution de ce sous–problème. Une partie du travail décrit ici a fait l'objet de publications [42, 8].

5.1 Description du problème

Dans cette section, nous commençons par décrire le problème de manière informelle, puis nous présentons les différentes notations utilisées pour les données du problème. Nous concluons par une formulation mathématique du TIST dans laquelle les interventions sous–traitées ne sont pas prises en compte.

5.1.1 Description générale

Le problème traite d'interventions devant être affectées à des équipes de techniciens. Les techniciens sont caractérisés par leurs jours de congés et leurs niveaux de compétence, et les interventions par leur priorité, leur temps d'exécution, leur liste de prédécesseurs (interventions qui doivent être traitées avant l'intervention) et le nombre de techniciens requis pour chaque niveau et domaine de compétence. L'objectif consiste à construire des équipes de techniciens pour chaque journée et à affecter des interventions à ces équipes en respectant toutes les contraintes de la planification et en minimisant la fonction objective

$$28t_1 + 14t_2 + 4t_3 + t_4$$

où t_k est la date de fin de la dernière intervention de priorité k pour $k = 1, 2, 3$ et t_4 est la date de fin de l'ordonnancement.

Un ordonnancement doit satisfaire une liste de contraintes pour l'affectation des techniciens et pour l'affectation des interventions. Nous considérons que chaque journée de travail est dans l'intervalle de temps $[0, H_{\max}]$ et qu'il est impératif de respecter cet intervalle. En conséquence, une intervention ne peut être réalisée avant la date 0 ou après la date $H_{\max}$ et ne peut être réalisée sur plusieurs jours.

Une intervention doit être réalisée par une seule équipe à une unique date, plusieurs équipes ne peuvent se partager la même intervention et aucune intervention ne peut être affectée à un technicien en repos. Une contrainte forte est que les équipes ne peuvent changer durant la journée, ce qui implique que chaque technicien appartient au plus à une seule équipe chaque jour. Cette contrainte est due au nombre limité de voitures disponibles et au temps que cela prendrait de rapatrier les voitures pour former de nouvelles équipes.

Une équipe doit satisfaire les demandes de chaque intervention qui lui est assignée. Ainsi, pour chaque intervention, nous devons affecter suffisamment de techniciens qualifiés pour satisfaire toutes les demandes. Par exemple, une intervention qui nécessite un technicien de niveau 2 dans le domaine d1 peut être réalisée par un technicien de niveau 2, 3 ou 4 dans le domaine d1, mais ne peut être réalisée par deux techniciens de niveau 1 dans le domaine d1. Le nombre requis de techniciens d'un certain niveau pour une intervention est cumulable puisqu'un technicien d'un niveau donné est aussi qualifié pour tous les niveaux inférieurs pour le même domaine de compétence.

Finalement, il est possible de sous–traiter certaines interventions à une entreprise externe. Chaque intervention a un coût spécifique dans le cas où elle serait sous–traitée et le coût total de la sous–traitance ne peut excéder un budget donné. Notons que le modèle mathématique que nous exposons en section 5.1.2 ne prend pas en compte les interventions sous–traitées. En effet, cette partie du problème est réalisée dans une phase de prétraitement décrite en section 5.2.1.

La méthode GRASP consiste à alterner itérativement une phase de construction et une phase d'amélioration. La phase de construction génère une solution réalisable en insérant des interventions suivant un critère d'insertion, le voisinage de cette solution est ensuite exploré avec un algorithme de recherche locale dans la phase d'amélioration jusqu'à ce qu'un minimum local soit identifié. Un des inconvénients de l'implémentation classique de la méthode GRASP est qu'elle ne prend pas en compte l'information fournie par les solutions précédemment explorées car chaque itération est indépendante de l'autre. Nous proposons une implémentation intégrant l'apprentissage[2] dans le but de diriger la recherche vers des solutions de qualité. En effet, à chaque itération, les critères d'insertion utilisés pour construire une solution réalisable sont mis à jour en prenant en compte les explorations passées.

[2]Une bibliographie sur d'autres versions de GRASP intégrant l'apprentissage peut être trouvée dans l'article de Pitsoulis et Resende [57].

Notre approche est centrée sur trois phases principales : une phase de prétraitement qui sélectionne un sous–ensemble d'interventions à sous–traiter ; une phase d'initialisation qui cherche à identifier de bons critères d'insertion pour la phase de construction ; et une phase de recherche qui utilise GRASP pour trouver la meilleure solution possible.

5.1.2 Notations et modèle mathématique

Dans cette section, nous introduisons un ensemble de notations ainsi que le modèle mathématique du problème.

Les constantes du problème sont les suivantes :

- $H_{\max}$ est la durée de temps de chaque jour ($H_{\max} = 120$ dans le sujet).
- $T(I)$ est le temps d'exécution de l'intervention I.
- $cost(I)$ est le coût de l'intervention I.
- A est le budget alloué pour les interventions sous–traitées.
- $P(t, j)$ est égale à 1 si le technicien t travaille le jour j, 0 sinon.
- $C(t, i)$ est le niveau de compétence du technicien t dans le domaine i.
- $R(I, i, n)$ est le nombre de techniciens requis de niveau n dans le domaine i pour traiter l'intervention I.
- $Pred(I)$ est la liste des prédecesseurs de I.
- $Succ(I)$ est la liste des successeurs de I.

Les variables sont les suivantes :

- $s(I)$ est la date de début de l'intervention I.
- $e(t, j)$ est le numéro de l'équipe à laquelle est rattaché le technicien t pour le jour j. L'équipe 0 est une équipe spécifique composée des techniciens ne travaillant pas ce jour.
- $d(I)$ est le jour où l'intervention I est planifiée.

Une intervention qui nécessite, pour le domaine i, au moins un technicien de niveau 3 et un technicien de niveau 2 aura ses demandes notées : $R(I, i, 1) = 2$, $R(I, i, 2) = 2$, $R(I, i, 3) = 1$ et $R(I, i, 4) = 0$.

Pour le modèle mathématique, nous utilisons les constantes suivantes :

- $Pr(k, I)$ égale 1 si la priorité de l'intervention I est k et 0 sinon.
- $\mathcal{P}(I_1, I_2)$ égale 1 si l'intervention I_1 est un prédécesseur de l'intervention I_2 et 0 sinon.

et les variables suivantes :

- $x(I, j, h, \epsilon)$ égale 1 si l'équipe ϵ travaille sur l'intervention I le jour j à la date de début h et 0 sinon.
- $y(j, \epsilon, t)$ égale 1 si le technicien t est dans l'équipe ϵ le jour j et 0 sinon.
- t_k, $k = 1, 2, 3$ est la date de fin de la dernière intervention de priorité k.
- t_4 est la date de fin de l'ordonnancement.

Ce problème, noté (P'), peut être modélisé de la manière suivante :

$$\text{Minimiser} \quad 28t_1 + 14t_2 + 4t_3 + t_4$$

$$\text{sujet à} \quad \sum_{j,h,\epsilon} x(I,j,h,\epsilon) = 1, \qquad \forall I, \tag{5.1}$$

$$y(j,0,t) = 1 - P(t,j), \qquad \forall j,t, \tag{5.2}$$

$$\sum_{\epsilon} y(j,\epsilon,t) = 1, \qquad \forall j,t, \tag{5.3}$$

$$x(I,j,h,0) = 0, \qquad \forall I,j,h, \tag{5.4}$$

$$\sum_{h_1=\max(h_2-T(I_1)+1,0)}^{\min(h_2+T(I_2)-1,H_{\max})} x(I_1,j,h_1,\epsilon) + x(I_2,j,h_2,\epsilon) \leq 1, \quad \forall I_1,I_2,h_2,j,\epsilon, \tag{5.5}$$

$$\sum_{j,h,\epsilon} (jH_{\max}+h)\,(x(I_1,j,h,\epsilon) - x(I_2,j,h,\epsilon)) + T(I_1)x(I_1,j,h,\epsilon) \leq 0,$$

$$\forall I_1,I_2 \mid \mathcal{P}(I_1,I_2) = 1, \tag{5.6}$$

$$x(I,j,h,\epsilon) = 0 \qquad \forall I,j,h,\epsilon \mid h + T(I) > H_{\max}, \tag{5.7}$$

$$\sum_{h} R(I,i,n)x(I,j,h,\epsilon) \leq \sum_{t|C(t,i)\geq n} y(j,\epsilon,t), \qquad \forall I,i,n,\epsilon,j, \tag{5.8}$$

$$\sum_{j,h,\epsilon} (jH_{\max}+h+T(I))\,Pr(k,I)x(I,j,h,\epsilon) \leq t_k, \quad \forall I, k = 1,2,3, \tag{5.9}$$

$$\sum_{j,h,\epsilon} (jH_{\max}+h+T(I))\,x(I,j,h,\epsilon) \leq t_4, \qquad \forall I. \tag{5.10}$$

La contrainte (5.1) assure que chaque intervention est traitée par une seule équipe à une journée et à une date fixée. La contrainte (5.2) garantit que si un technicien t ne travaille pas le jour j, il est dans l'équipe 0. La contrainte (5.3) spécifie qu'un technicien appartient à une seule équipe chaque jour. La contrainte (5.4) assure qu'aucune intervention n'est réalisée par l'équipe 0. La contrainte (5.5) certifie que deux interventions réalisées le même jour par la même équipe sont effectuées à des dates différentes. La contrainte (5.6) spécifie que tous les prédécesseurs d'une intervention donnée doivent être réalisés avant la date de début de l'intervention. La contrainte (5.7) assure que chaque jour a une durée limite de $H_{\max}$, nombre maximal de portions de temps par jour. La contrainte (5.8) spécifie qu'une équipe qui travaille sur l'intervention I satisfait les demandes en nombre de techniciens par niveau de compétence. Finalement, la contrainte (5.9) spécifie que t_k est la date de fin de la dernière intervention de priorité k, $k = 1,2,3$ et la contrainte (5.10) spécifie que t_4 est la date de fin de l'ordonnancement.

5.2 Méthode de résolution

Dans cette section, nous détaillons les trois phases principales de notre approche : (i) la phase de prétraitement qui consiste à sélectionner un sous–ensemble d'interventions à sous–traiter ; (ii) la phase d'initialisation qui cherche à identifier de bons critères d'insertions pour la construction des solutions ; et (iii) la phase de recherche qui s'appuie sur la métaheuristique GRASP pour trouver la meilleure solution possible. La terminologie GRASP se réfère à une classe de procédures dans laquelle une heuristique gloutonne et une technique de recherche locale sont employées. La métaheuristique GRASP a été appliquée à de nombreux problèmes d'optimisation combinatoire comme les problèmes d'ordonnancement [75], les problèmes de routage [1], les problèmes de graphes [59], les problèmes d'affectation [23], les problèmes de planification [75] etc. On peut se référer à [**?**] pour une bibliographie plus complète sur le sujet.

La méthode GRASP consiste à répéter la procédure suivante jusqu'à satisfaire un critère d'arrêt (nombre maximum d'itérations, temps CPU fixé, niveau de qualité de la solution...) :

1. Générer une solution réalisable avec un algorithme glouton randomisé.
2. Appliquer un algorithme de recherche locale à la solution précédente.
3. Mettre à jour la meilleure solution.

5.2.1 Heuristique de choix des interventions à sous–traiter

Dans le contexte du challenge, nous devions trouver une méthode efficace dans un temps limité. Pour cela, nous avons opté pour une méthode heuristique qui consiste à choisir un ensemble d'interventions à sous–traiter au début du processus. Une fois cet ensemble déterminé, les interventions sous–traitées ne sont plus prises en compte durant le processus de recherche. Cette heuristique est fondée sur un critère d'insertion lié à un poids spécifique attribué à chaque intervention. Ce poids $\omega(I)$ est établi à partir d'une borne supérieure du nombre minimum de techniciens nécessaires à la réalisation de l'intervention I ($\texttt{mintec}(I)$), et de la durée de celle-ci ($T(I)$). Il est égal au produit de ces deux valeurs : $\omega(I) = \texttt{mintec}(I) \times T(I)$.

Soit Ω_t l'ensemble des indices des variables représentant les techniciens et $x \in \{0,1\}^{|\Omega_t|}$ un vecteur de variables de décision. Dans un premier temps, la valeur $\texttt{mintec}(I)$ est calculée en résolvant de manière heuristique le programme linéaire en nombres entiers suivant :

$$\begin{aligned} \text{Minimiser} \quad & \sum_{t \in \Omega_\text{t}} x_t \\ \text{sujet à} \quad & \sum_{t|C(t,I) \geq n, t \in \Omega_\text{t}} x_t \geq R(I,i,n), \quad \forall i,n, \\ & x_t \in \{0,1\}, \quad t \in \Omega_\text{t}. \end{aligned}$$

L'heuristique utilisée pour résoudre ce problème est décrite par l'algorithme 5.1.