Propagation de Critères Relatifs à l'Acceptation Pratique des Solutions

Thierry Petit

thierry.petit@emn.fr

GOThA - Contraintes et R.O. - 5 juin 2007

Contexte

Problèmes Sur-Contraints

- Très fréquents en pratique
- Contraintes complexes à imposer pour conserver des solutions acceptables en pratique
- **Problème d'optimisation à contraintes :** comment obtenir une résolution *robuste*, si possible avec les outils existants ?

Plan de l'Exposé

- Problèmes Sur-Contraints
- Problème d'ordonnancement avec dépassements de capacité
- La contrainte SoftCumulative
- Propagation de contraintes relatives à l'acceptation des solutions
- Expérimentations

Problèmes Sur-Contraints

Problèmes Sur-Contraints

- Contraintes physiques (ou « dures »)
- Contraintes molles (ou « préférences »)

Problèmes Sur-Contraints (2)

- Problèmes d'optimisation classiques

  Peuvent être modélisés et résolus directement dans le cadre de la PPC

  [Petit, Régin, Bessière, ICTAI'00]

- Très fréquents en pratique

  Cahiers des charges :

  - Écrits par des non-informaticiens
  - Toutes les contraintes, importantes ou moins importantes, sont au même niveau

Exemple (projet IST)

Approches Standard

Relaxation

- On relâche les contraintes, en associant à chacune d'entre elles un « coût de violation » marquant la distance à l'état « C satisfaite »
- On résout **un problème d'optimisation** relatif aux coûts de violation

Approches Standard

Relaxation

• On relâche les contraintes, en associant à chacune d'entre elles un « coût de violation » marquant la distance à l'état « C satisfaite »

• On résout **un problème d'optimisation** relatif aux coûts de violation

Décomposition

• On gère à la main le problème, en ajoutant/supprimant des contraintes

• On résout dynamiquement **plusieurs problèmes de satisfaction (*explications*)**

Approches Standard (2)

Relaxation
Algorithmes dédiés « académiques » efficaces mais il est parfois difficile de caractériser la qualité des solutions (affecter un coût à la violation de chaque contrainte n'est pas suffisant en pratique)

Décomposition
Approche plus réaliste mais empirique (peu d'algorithmes dédiés)

Approches Standard (2)

Relaxation
Algorithmes dédiés
« académiques » efficaces
mais il est parfois difficile
de caractériser la qualité
des solutions (affecter un
coût à la violation de
chaque contrainte n'est
pas suffisant en pratique)

Décomposition
Approche plus réaliste
mais trop empirique (peu
d'algorithmes dédiés)

Problèmes réels ?
Nécessité d'intégrer
des techniques de relaxation
aux solveurs pour bénéficier
d'algorithmes dédiés

Relaxation

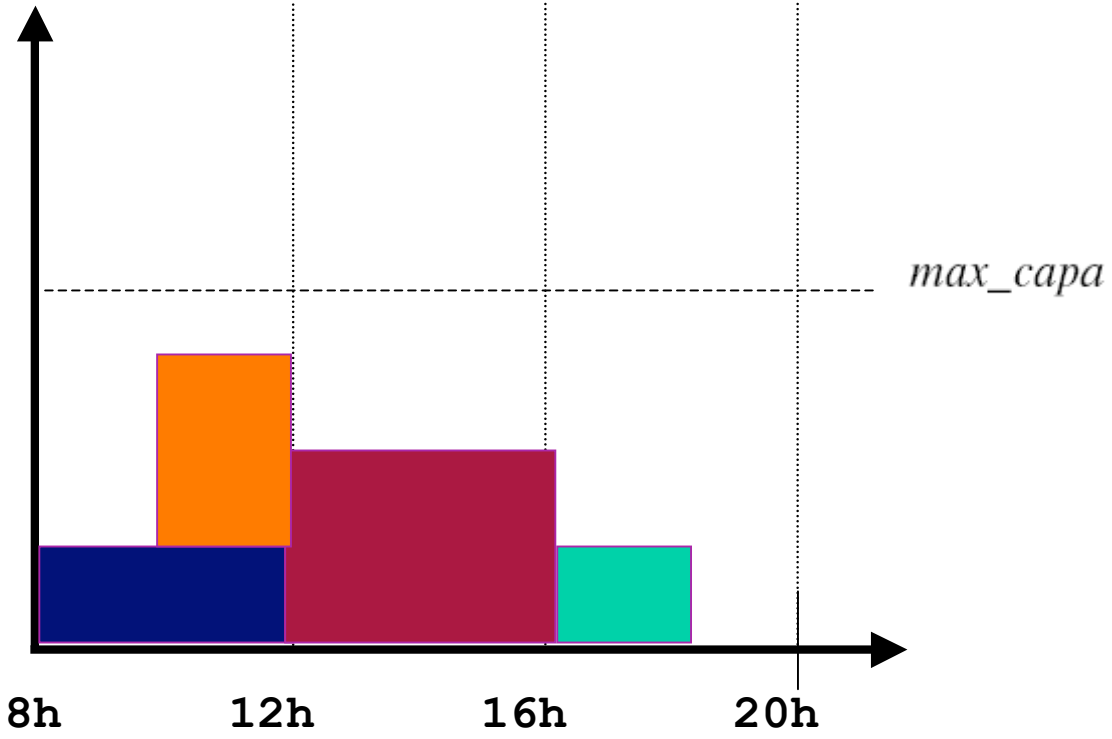
Problèmes sur-contraints réels : les solutions doivent respecter des **règles sur les violations**

- *Ne pas violer de contrainte physique*
- *Favoriser la satisfaction des contraintes importantes*
- *Limiter le « degré de violation » des contraintes*
- *Contrôler la répartition topologique des violations dans le réseau de contraintes*
- *Exprimer des dépendances entre violations (règles « métier »)*

Problème d'Ordonnancement avec Dépassements de Capacité

Contrainte Cumulative

[Aggoun and Beldiceanu 93]



Profil de consommation de ressource

Problèmes à Ressource Humaine

- La ressource est l'équipe d'employés dont on dispose
- Quand le problème est sur-contraint, on peut autoriser des dépassements de capacité
  - Embauche / heures supplémentaires
  - Réalisation d'une tâche avec moins de personnel que prévu

Cas d'Étude

– Emploi du temps découpé en demi-journées (4h)
– Durées fixées (≤ 4h)
– Précédences entre les tâches
– Nombre réel d'employés : *real_capa*
– **Contraintes d 'acceptabilité des solutions**

Cas d'Étude (2)

Contraintes d 'acceptabilité des solutions :

- ***Borner*** *les dépassements de ressource à chaque point de temps (1h) ,*
- ***Minimiser*** *leur somme,*
- ***Répartir*** *de façon homogène les dépassements,*
- ***Règle métier*** *: Si en fin de demi-journée on a un dépassement alors la première heure de la demi-journée suivante ne doit pas subir un dépassement*

Modèle à Contraintes

Violation = dépassement à un point de temps

Modèle à Contraintes (2)

Deux alternatives :

(1) Gérer les violations par une structure externe

(2) Ajouter des variables de coût au problème

Modèle à Contraintes (2)

Deux alternatives :

(1) Gérer les violations par une structure externe

(2) Ajouter des variables de coût au problème

(1) Contraintes d'acceptabilité des solutions modélisées via une fonction objectif

Modèle à Contraintes (2)

Deux alternatives :

(1) Gérer les violations par une structure externe

(2) Ajouter des variables de coût au problème

(1) Contraintes d'acceptabilité des solutions modélisées via une fonction objectif

(2) Contraintes d'acceptabilité des solutions modélisées via des contraintes (globales)

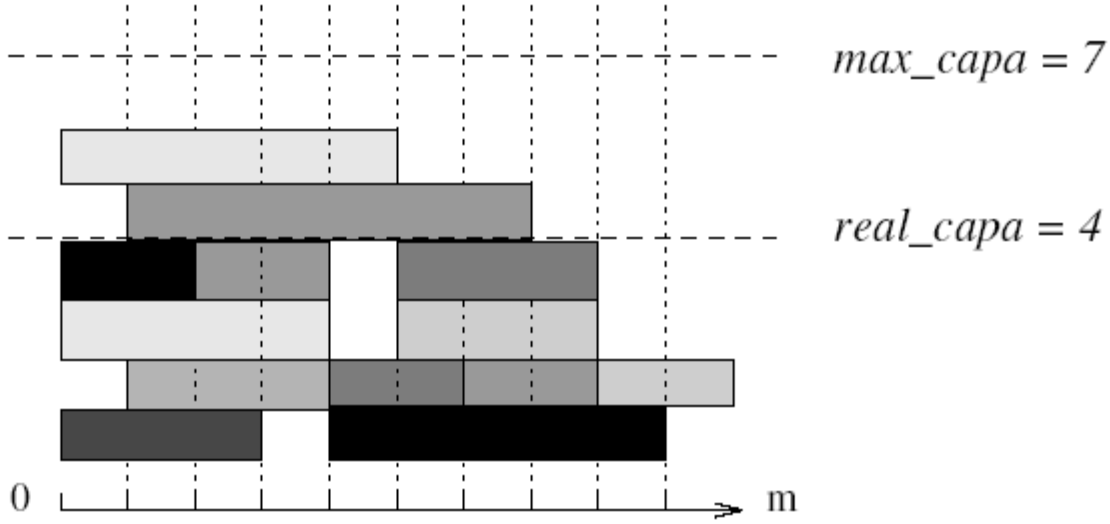
Modèle à Contraintes (3)

(2) Contraintes d'acceptabilité des solutions modélisées via des contraintes (globales)

Propagation intéressante ?

La Contrainte Soft-Cumulative

Principe



$cost = 0 + 2 + 2 + 1 + 0 + 1 + 1 + 0 + 0 + 0 = 7$

Cas particulier où chaque tâche consomme 1 ressource

Définition

Definition 3. *Consider one resource and a set of jobs scheduled between time* 0 *and* m*, each job consuming a positive amount of resource. At each point in time,* $real\_capa$ *and* max_capa *are two limits of resource, s.t.* $real\_capa \leq max\_capa$*.* $cost$ *is an integer variable. The* `SoftCumulative` *constraint enforces that:*

- *At each point in time* i*, the cumulated height* h_i *(sum of amounts of resource) of the set of jobs that overlap* i*, does not exceed* max_capa*.*
- *For each job* j *the constraint* $starts[j] + d_j = ends[j]$ *is satisfied.*
- *The following constraint holds:*

$$cost = \sum_{i \in \{0, m-1\}} max(0, h_i - real\_capa)$$

Filtrage Indépendant de la Relaxation

- La contrainte SoftCumulative définit implicitement une contrainte Cumulative telle que la capacité maximale soit *max_capa*
- Filtrage utilisé : « sweep » [Beldiceanu and Carlsson 2003]

Filtrage lié à la Relaxation

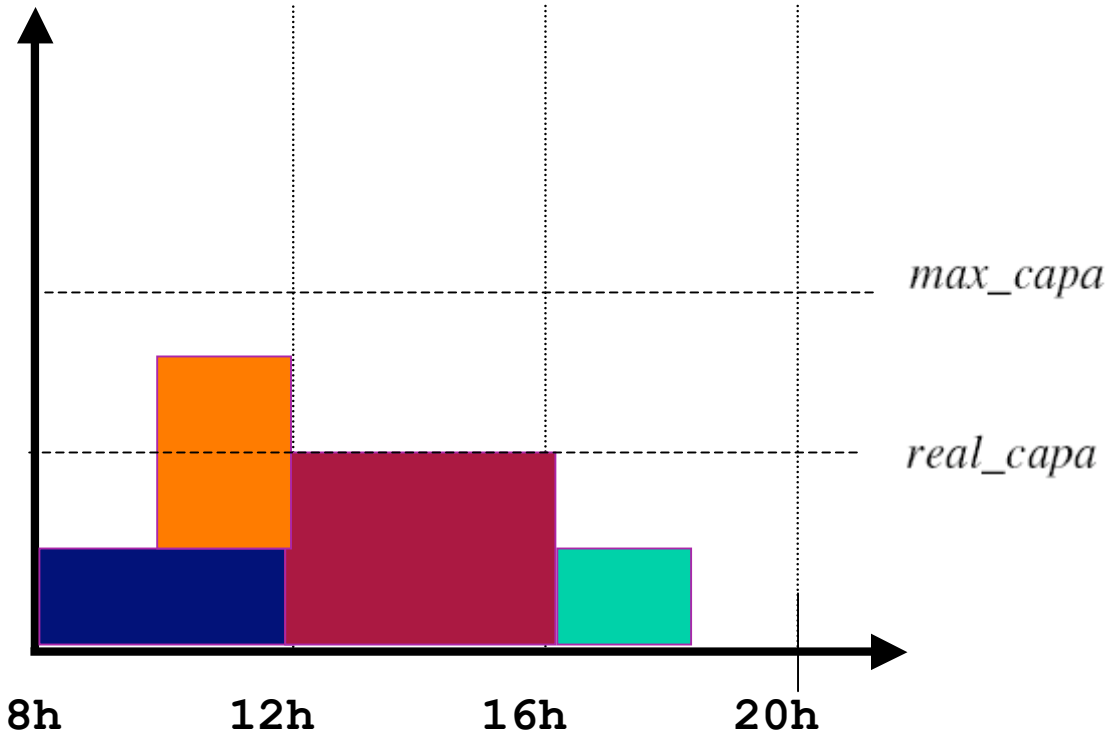
- Variables

```
IntDomainVar[] starts;   // array of start times of jobs
IntDomainVar[] costVars; // array of over-loads at each point in time
IntDomainVar cost;       // global cost of violation of the constraint
```

- propagation « standard »
  - **`starts[] => cost`**
  - **`cost => starts[]`**
- propagation issue des variables de coût
  - **`costVars[] => starts[]`**
  - **`cost => costVars[]`**

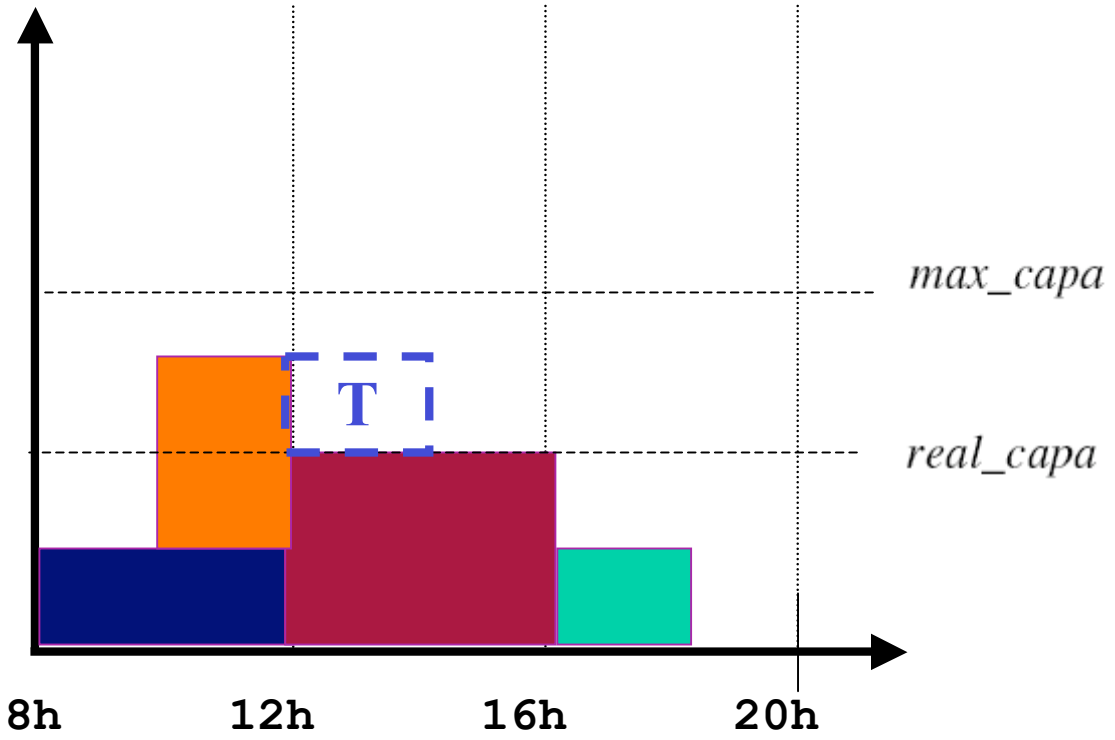
Propagation « Standard »

`distance` = dépassement du profil courant au dessus de real_capa (tâches déjà fixées)



Un Premier Filtrage

`delta[T,12]` = augmentation de la `distance` si T est placée à 12 heures



Un premier Filtrage (2)

Si `distance + delta[T,12] > max(cost)` alors
12h n'est pas une valeur valide pour `starts[T]`

Un premier Filtrage (3)

Amélioration : introduction d'une **borne inférieure globale LB**, sous-estimant le dépassement induit nécessairement par toutes les tâches non fixées sauf `T`

Filtrage avec Borne Inférieure

Si `distance + delta[T,12] + LB > max(cost)`
alors 12h n'est pas une valeur valide pour `starts[T]`

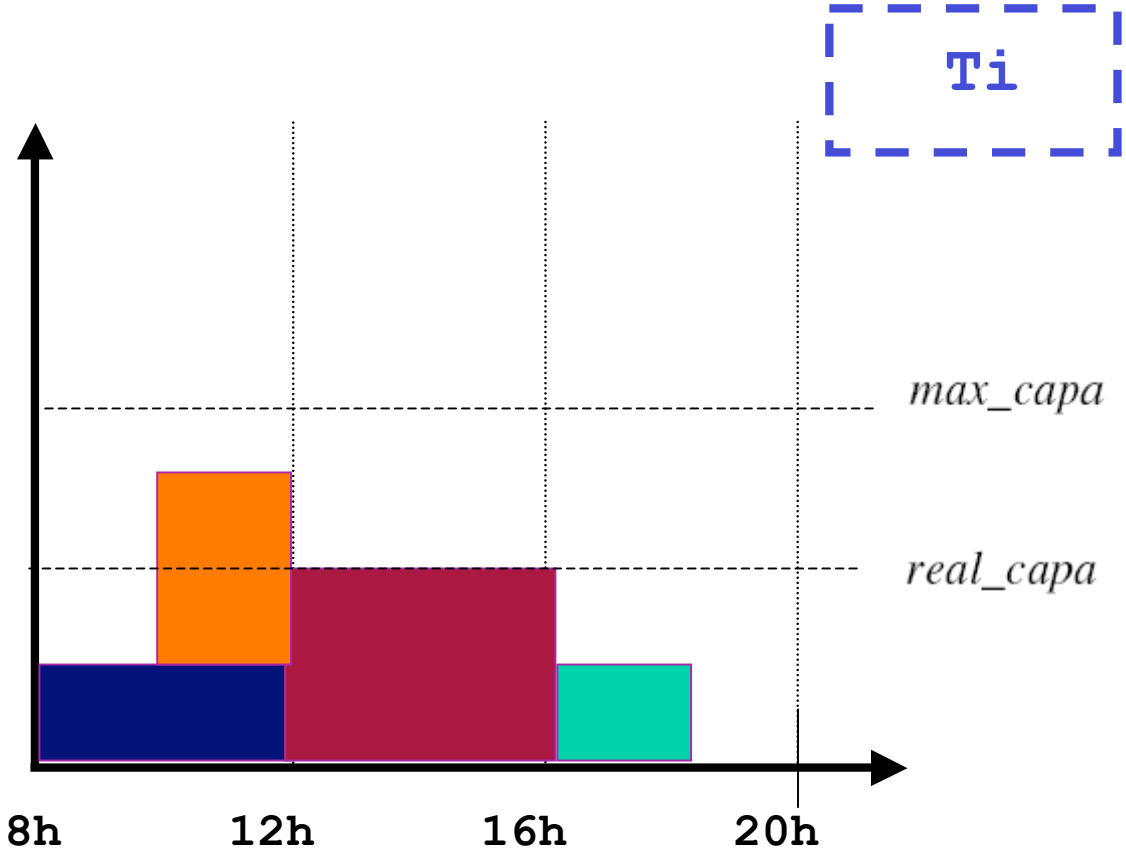
Filtrage avec Borne Inférieure

Si `distance + delta[T,12] + LB > max(cost)`
alors 12h n'est pas une valeur valide pour `starts[T]`

- LB1 : agrégation de violations locales
- LB2 : parties obligatoires (P.O),
- LB3 : P.O. + surface requise + intervalles

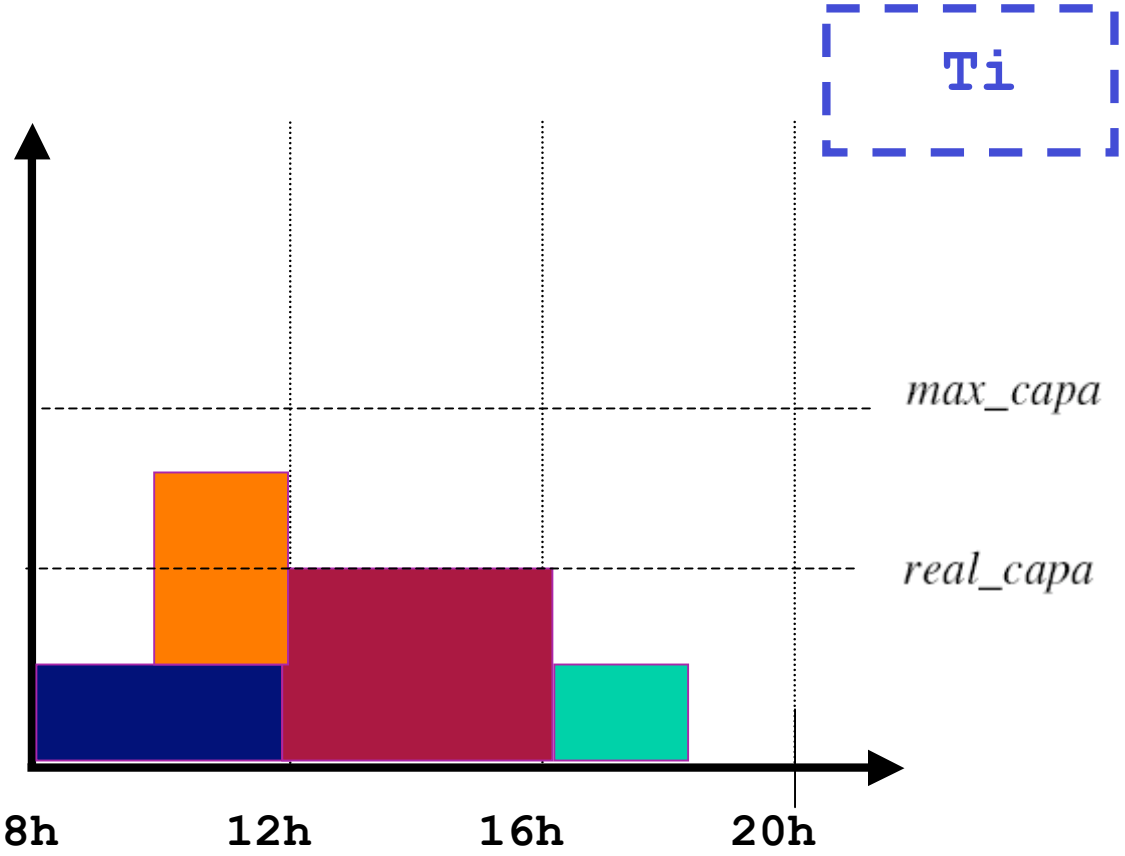
Agrégation de Violations Locales

Où qu'on la place, `Ti` engendrera un dépassement `#inc(Ti)` en plus



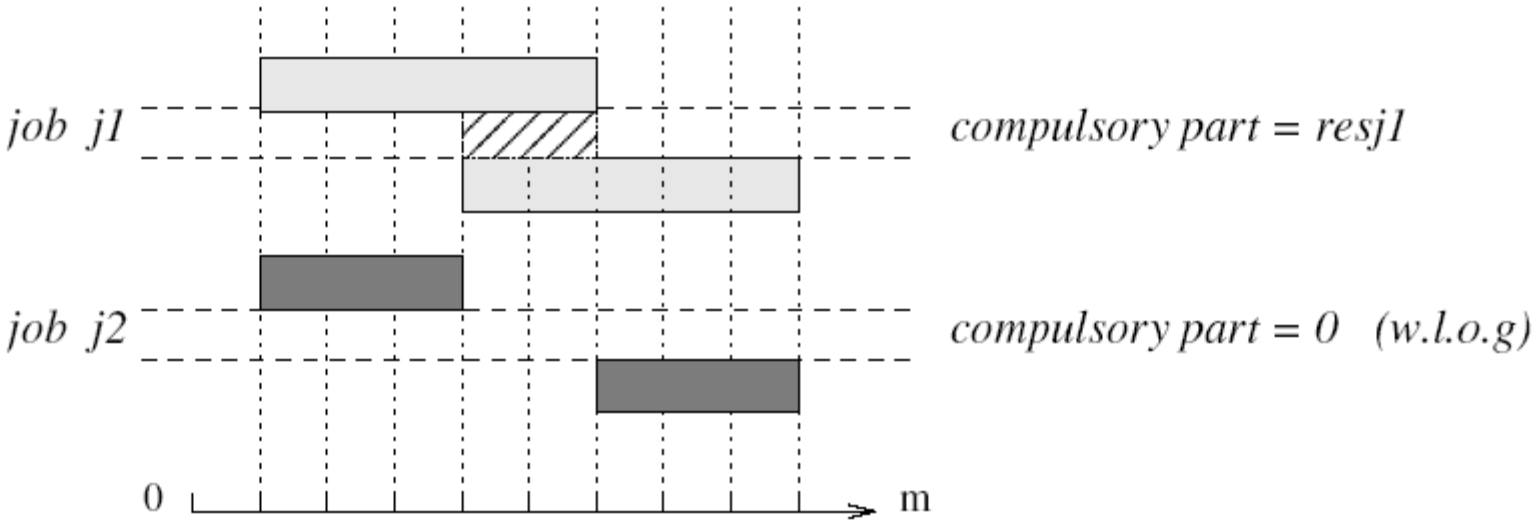
Agrégation de Violations Locales

Où qu'on la place, `Ti` engendrera un dépassement `#inc(Ti)` en plus

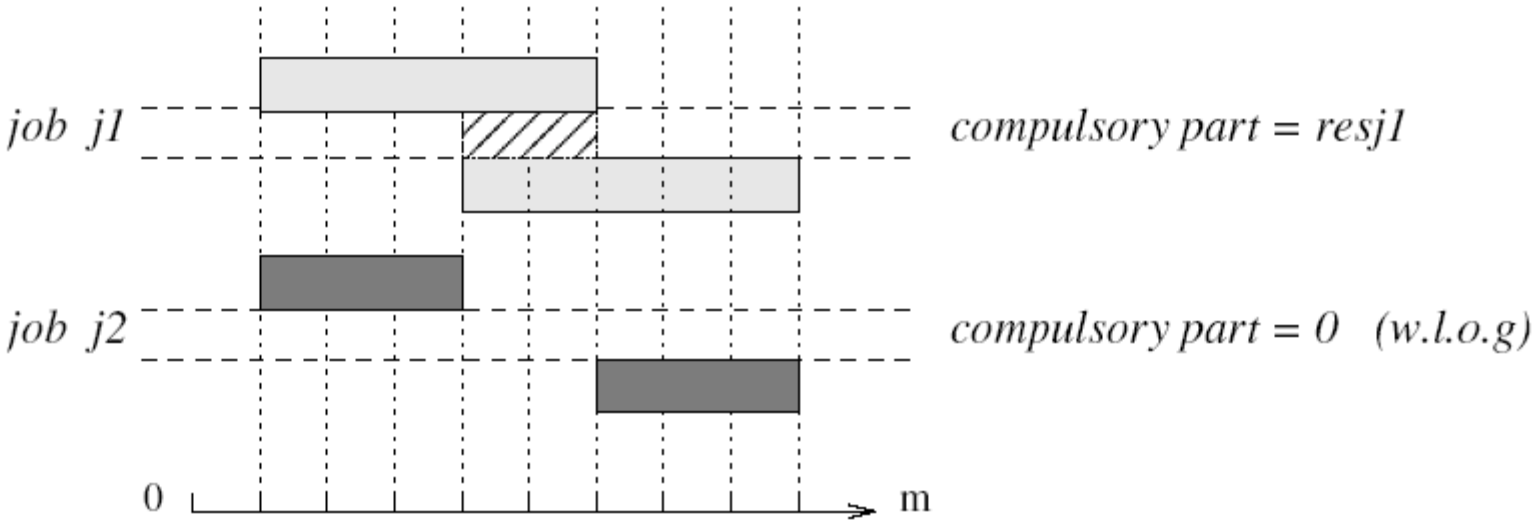


$$LB1 = \Sigma\ \#inc(Ti)$$

Partie Obligatoires

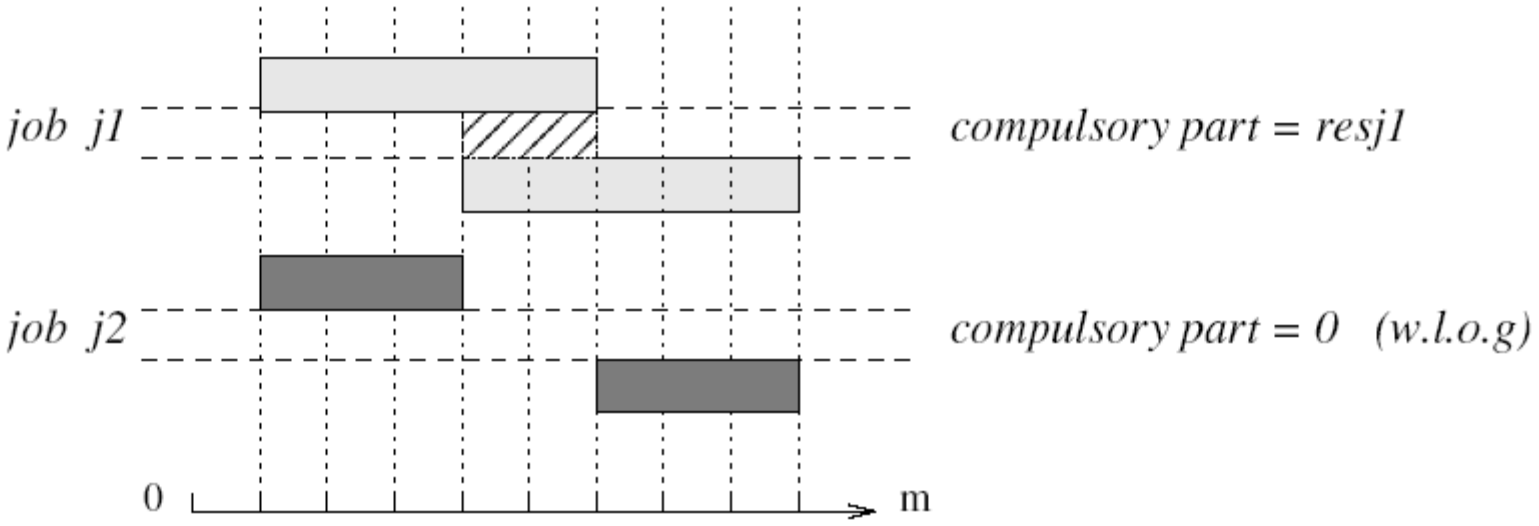


Partie Obligatoires



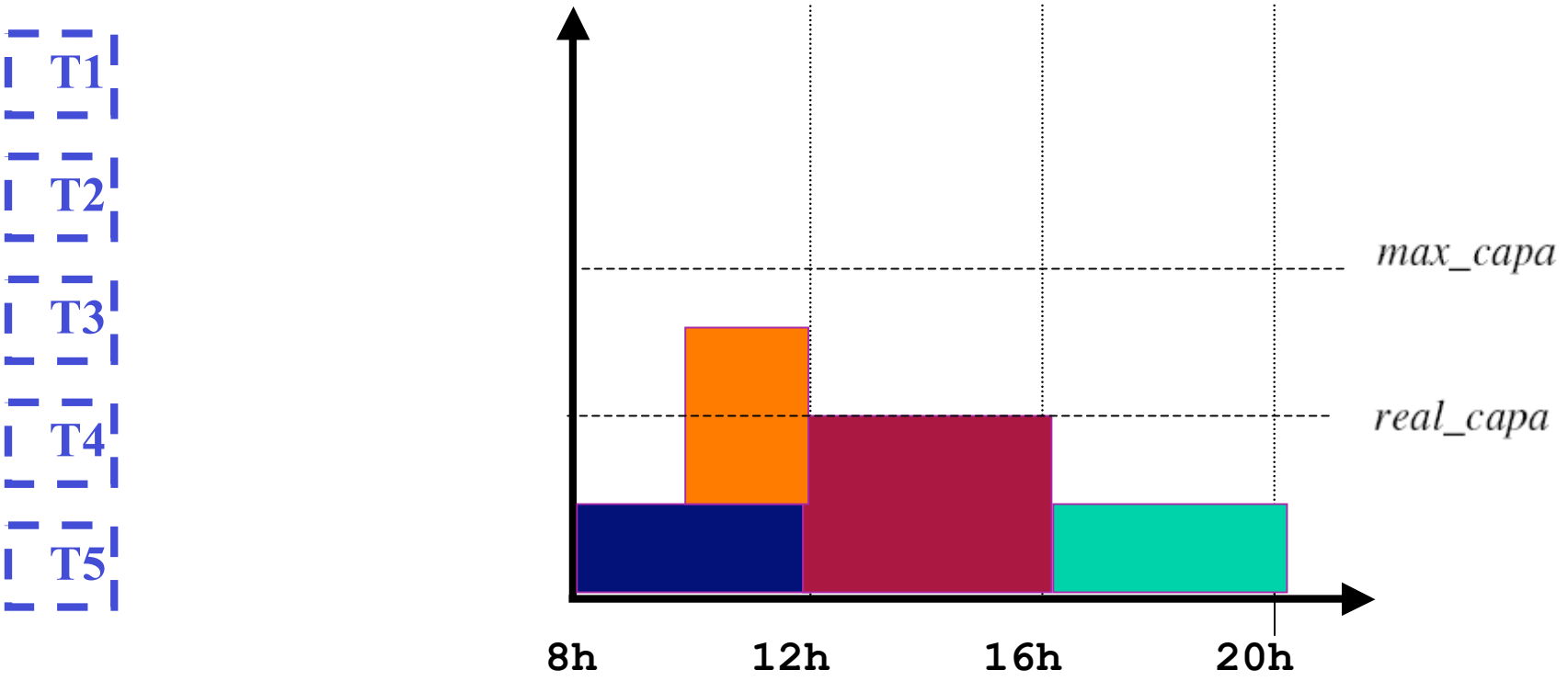
On peut augmenter le profil avec les parties obligatoires en les considérant comme des tâches fixées, et améliorer `distance`

Partie Obligatoires



LB2 = distance(profil + PO)

Surface Requise



Surface Requise

`LB3 = distance(profil + PO)`
+ dépassement global
en surface des tâches
à placer moins leurs parties obligatoires

Prise en Compte des Domaines des Tâches

- Chaque tâche n’a pas un début possible à 0 et une fin possible à l ’horizon
- On perd donc à évaluer la surface globale de 0 à l’horizon

Idée : classer les tâches par débuts croissants et fins décroissantes, et considérer des intervalles formés par ces deux items.

Prise en Compte des Domaines des Tâches (2)

- Toute borne **LB3+** obtenue avec les tâches strictement incluse dans un intervalle donné peut remplacer **LB3**
- De plus, les bornes issues d'intervalles disjoints peuvent être sommées (tâches disjointes)

Combinaison des Bornes ?

- `LB1` et `LB2,3` sont (trivialement) non comparables ...

Propagation Issue des Variables de Coût (costVars[ ] => starts[ ])

Test de consistance des débuts en fonction des bornes des costVars[ ] impliquées

For any reason $max(costVars[i])$ can be reduced. In this case, $max(costVars[i])$ is strictly less than max_capa and we can apply the following corollary.

Corollary 5 *Let $j \in U$ and $i \in D(start[j])$ s.t. i is not in $[max(D(start[j]), min(D(end[j]))[$ (corresponding to compulsory part). If $res_j + h_i(\mathcal{F}) + h_i(\mathcal{U}_{CP}) > max(costVars[i])$ then i can be removed from $D(start[j])$.*

Back-Propagation de l'objectif Global (cost => costVars[ ])

Filtrage usuel de la contrainte somme

Propagation de Contraintes Relatives à l'Acceptation des Solutions

Modèle

```
// Data
   int[] ds;  // durations
   int[] hs;  // heights
   int nPrec; // number of precedence constraints (if any)
   int real_capa, max_capa;
// Variables
   IntDomainVar[] starts;
   IntDomainVar[] ends;
   IntDomainVar[] costVars;
   IntDomainVar cost;
// Constraints
   SoftCumulative(starts, ends, costVars, cost, // R.1 and R.2
                  ds, hs, real_capa, max_capa);
   For(i:1..nPrec): two jobs j1, j2
      [starts[j1]+ds[j1] <= starts[j2]]; // random precedences
   For each array half-day, element of a partition of costVars[]: // R.3
      AtLeast(half-day,2,0));             // at least 2 costVars[i]==0
      AtMost(half-day,2,1));              // at most 2 costVars[i]==1
      AtMostKNotZeroOrOne(half-day,1));  // at most 1 costVars[i]>1
   For(i:4..costVars.length-1):           // R.4
      if(i%4==0): [costVars[i-1]==0 || costVars[i]==0];
// Objective
   minimize(cost)
```

Modèle

```
// Data
   int[] ds;  // durations
   int[] hs;  // heights
   int nPrec; // number of precedence constraints (if any)
   int real_capa, max_capa;
// Variables
   IntDomainVar[] starts;
   IntDomainVar[] ends;
   IntDomainVar[] costVars;
   IntDomainVar cost;
// Constraints
   SoftCumulative(starts, ends, costVars, cost, // R.1 and R.2
                  ds, hs, real_capa, max_capa);
   For(i:1..nPrec): two jobs j1, j2
      [starts[j1]+ds[j1] <= starts[j2]]; // random precedences
   For each array half-day, element of a partition of costVars[]: // R.3
      AtLeast(half-day,2,0));              // at least 2 costVars[i]==0
      AtMost(half-day,2,1));               // at most 2 costVars[i]==1
      AtMostKNotZeroOrOne(half-day,1));  // at most 1 costVars[i]>1
   For(i:4..costVars.length-1):            // R.4
      if(i%4==0): [costVars[i-1]==0 || costVars[i]==0];
// Objective
   minimize(cost)
```

Modèle (2)

- Cas 1: Contraintes de répartition : GCC (global cardinality constraints)
- Cas 2 : Contraintes de répartition : primitives réagissant aux instantiations des variables de coût

Modèle (2)

- Cas 1: Contraintes de répartition : GCC (global cardinality constraints)
- Cas 2 : Contraintes de répartition : primitives réagissant aux instantiations des variables de coût

Résultats

Recherche de l'Optimal

Choix des instances les plus difficiles pour le filtrage basé sur les surfaces (côut global de violation proche de zéro)

Instance	*cost*	Complete Model	Uncomplete Model
1	2	182 (241 ms)	611 (70 ms)
2	2	2717 (1,27 s)	23304 (17,8 s)
3	0	115 (10 ms)	3390 (120 ms)
4	2	1726 (641 ms)	5292 (731 ms)
5	0	63 (0 ms)	83 (10 ms)
6	0	56 (0 ms)	76 (10 ms)
7	4	1858 (561 ms)	6016 (922 ms)

m = 12

Recherche de l'Optimal (2)

Choix des instances les plus difficiles pour le filtrage basé sur les surfaces (côut global de violation proche de zéro)

Instance	*cost*	Complete Model	Uncomplete
1	3	81834 (53s)	– (> 5 min)
2	1	22966 (17,9 s)	– (> 5 min)
3	4	59096 (35,7 s)	– (> 5 min)
4	2	18166 (13,2 s)	– (> 5 min)
5	1	5934 (5 s)	– (> 5 min)

m = 16

Taille Réelle

- Problème de décision imposant sur une semaine (m = 36) qu'au plus un total correspondant à une unité de ressource de dépassement par jour soit tolérée (soit max(cost) = 9)
  - Pas de difficulté pour le modèle avec GCC (moins de 100 nœuds par instance)
  - Pas de solution sans propagation de la GCC