Modélisation avec des contraintes sur un domaine fini

- Domaines et "Labeling"
- Contraintes complexes
- Exemples
- voir livre Marriott/Stuckey

Domaines

- On doit définir les domaines des variables
- Si aucun domaine est donné, on prend un domaine prédéfini
- Sac du contrebandier
  $D(W) = [0..9]$, $D(P) = [0..9]$, $D(C) = [0..9]$

```
?- fd_domain([W,P,C],0,9), 4*W + 3*P + 2*C #=< 9,
                               15*W + 10*P + 7*C #>= 30.
C = _#47(0..4)
P = _#25(0..3)
W = _#3(0..2)
```

Labeling

- Un prédicat prédéfini lance le solutionneur.

```
?- fd_domain([W,P,C],0,9), 4*W + 3*P + 2*C #=< 9,
               15*W + 10*P + 7*C #>= 30,fd_labeling([W,P,C]).
C = 3
P = 1
W = 0 ? ;
C = 0
P = 3
W = 0 ? ;
C = 1
P = 1
W = 1 ? ;
C = 0
P = 0
W = 2
```

Optimisation

```
?- fd_domain([W,P,C],0,9),
   4*W + 3*P + 2*C #=< 9,
   15*W + 10*P + 7*C #= Max,
   fd_maximize(fd_labeling([W,P,C,Max]),Max).

C = 1
Max = 32
P = 1
W = 1
```

Contraintes complexes

- permettent de modéliser plus succinctement
- meilleure propagation, meilleure efficacité
- Exemples
  - `alldifferent`
  - `element(I, List, X)` contraint `X` d'être égale au `I`ème entier de la liste `List`.

    ```
    ?- fd_element(3,[1,2,4],E).

    E = 4
    ```

Exemple

- Problème d'affectation
- Quatre ouvriers w_1, w_2, w_3, w_4 et quatre produits p_1, p_2, p_3, p_4
- Affecter des ouvriers aux produits pour faire un profit ≥ 19
- Les profits sont donnés par

	p_1	p_2	p_3	p_4
w_1	7	1	3	4
w_2	8	2	5	1
w_3	4	3	7	2
w_4	3	1	6	3

1er modèle

- 16 variables booléennes B_{ij} qui signifient que ouvrier i est affecté au produit j
- $\bigwedge_{i=1}^{4} \sum_{j=1}^{4} B_{ij} = 1$
- $\bigwedge_{j=1}^{4} \sum_{i=1}^{4} B_{ij} = 1$
- $P = 7 * B_{11} + B_{12} + 3 * B_{13} + 4 * B_{14} + 8 * B_{21} + 2 * B_{22} + 5 * B_{23} + B_{24} + 4 * B_{31} + 3 * B_{32} + 7 * B_{33} + 2 * B_{34} + 3 * B_{41} + B_{42} + 6 * B_{43} + 3 * B_{44}$
- $P \geq 19$

2ème modèle

- Quatre variables correspondant aux ouvriers (quel domaine ?)
- Quatre variables correspondant aux profits par ouvrier (quel domaine ?)
- `alldifferent([W1,W2,W3,W4])`
- ```
  element(W1,[7,1,3,4],WP1)
  element(W2,[8,2,5,1],WP2)
  element(W3,[4,3,7,2],WP3)
  element(W4,[3,1,6,3],WP4)
  ```
- $P = WP1 + WP2 + WP3 + WP4, P \geq 19$
  ```

3ème modèle

- Quatre variables correspondant aux produits (quel domaine ?)
- Quatre variables correspondant aux profits (quel domaine ?)
- `alldifferent([T1,T2,T3,T4])`
- ```
  element(T1,[7,8,4,3],TP1)
  element(T2,[1,2,3,1],TP2)
  element(T3,[3,5,7,6],TP3)
  element(T4,[4,1,2,3],TP4)
  ```
- $P = TP1 + TP2 + TP3 + TP4, P \geq 19$
  ```

Quel est le meilleur modèle ?

- ça dépend

- Critères
  - Nombre de variables
  - Nombre de contraintes
  - Flexibilité
    - Comment ajouter la contrainte que on ne peut pas en même temps avoir ouvrier 1 affecté au produit 1 et ouvrier 4 affecté au produit 4 ?

Combiner les modèles

- Combiner les modèles en reliant les variables et leur valeurs dans chaque modèle

- p.e. $B_{13} = 1$ signifie $W1 = 3$ signifie $T_3 = 1$

- Les modèles combinés peuvent être plus efficace grâce à la propagation d'information

- On suppose qu'on a une contrainte `ssi(V1,D1,V2,D2)` qui est vraie, si (`V1=D1` si et seulement si `V2=D2`)

Modèle combiné

```
alldifferent([W1,W2,W3,W4])   alldifferent([T1,T2,T3,T4])
element(W1,[7,1,3,4],WP1)     element(T1,[7,8,4,3],TP1)
element(W2,[8,2,5,1],WP2)     element(T2,[1,2,3,1],TP2)
element(W3,[4,3,7,2],WP3)     element(T3,[3,5,7,6],TP3)
element(W4,[3,1,6,3],WP4)     element(T4,[4,1,2,3],TP4)
P #= WP1 + WP2 + WP3 + WP4    P #= TP1 + TP2 + TP3 + TP4
P #>= 19
ssi(W1,1,T1,1) ssi(W1,2,T2,1) ssi(W1,3,T3,1) ssi(W1,4,T4,1)
ssi(W2,1,T1,2) ssi(W2,2,T2,2) ssi(W2,3,T3,2) ssi(W2,4,T4,2)
ssi(W3,1,T1,3) ssi(W3,2,T2,3) ssi(W3,3,T3,3) ssi(W3,4,T4,3)
ssi(W4,1,T1,4) ssi(W4,2,T2,4) ssi(W4,3,T3,4) ssi(W4,4,T4,4)
```

Exemple: Ordonnancement

- Un ensemble de tâches est donné
  - avec des préséances (des tâches doivent être terminées avant des autres)
  - et des ressources partagées (des tâches ont besoin de la même machine)
- Déterminer un bon ordonnancement, donc
  - les contraintes sont satisfaites
  - le temps global est minimisé
- Ici nous fixons uniquement une limite.

Exemple

On représente les données comme une liste de tâches

```
task(nom,duree,[noms],machine)
```

```
problem([task(j1,3,[],m1),
 task(j2,8,[],m1),
 task(j3,8,[j4,j5],m1),
 task(j4,6,[],m2),
 task(j5,3,[j1],m2),
 task(j6,4,[j1],m2)]).
```

Programme

- Forme du programme
  - Définir les variables: `makejobs`
    - variables: Temps de début de chaque tâche
    - liste: `job(nom,duree,StartVar)`
  - Contraintes de préséances : `precedences`
  - Contraintes de machines : `machines`
  - Labeling : `labeltasks`
    - prendre des variables de la liste des jobs et donner une valeur

Programme

```
schedule(Data, End, Joblist)   :-
         makejoblist(Data, Joblist, End),
         precedences(Data, Joblist),
         machines(Data, Joblist),
         labeltasks(Joblist).

makejoblist([],[],_).
makejoblist([task(N,D,_,_)|Ts], [job(N,D,TS)|Js], End)  :-
        fd_domain(TS,0,1000),
        TS + D #=< End,
        makejoblist(Ts, Js, End).

getjob(JL, N, D, TS)  :- once(member(job(N,D,TS), JL)).
```

Programme: préséances

```
precedences([],_).
precedences([task(N,_,Pre,_)|Ts], Joblist)  :-
        getjob(Joblist, N, _, TS),
            prectask(Pre, TS, Joblist),
            precedences(Ts, Joblist).

prectask([],  _, _).
prectask([Name|Names], PostStart, Joblist)  :-
        getjob(Joblist, Name, D, TS),
            TS + D #=< PostStart,
            prectask(Names, PostStart, Joblist).
```

Programme: machines

```
machines([], _).
machines([task(N,_,_,M)|Ts], Joblist)  :-
        getjob(Joblist, N, D, TS),
            machtask(Ts, M, D, TS, Joblist),
            machines(Ts, Joblist).

machtask([], _, _, _, _).
machtask([task(SN,_,_,M0)|Ts], M, D, TS, Joblist)  :-
             (M = M0 ->
                  getjob(Joblist, SN, SD, STS),
                  exclude(D, TS, SD, STS)
             ; true ),
            machtask(Ts, M, D, TS, Joblist).

exclude(_D, TS, SD, STS)  :-  STS + SD #=< TS.
exclude(D, TS, _SD, STS)  :-  TS + D #=< STS.
```

Labeling

```
labeltasks([]).
labeltasks([job(_,_,TS)|Js])    :-
     fd_dom(TS,L),member(TS,L),
     labeltasks(Js).
```

Exécuter ordonnancement

```
?- problem(Problem), End = 20, schedule(Problem,End,LJobs).
```

`makejobs` : construire les contraintes initiales et ajouter des contraintes

```
[job(j1,3,TS1),job(j2,8,TS2),job(j3,8,TS3),
 job(j4,6,TS4),job(j5,3,TS5),job(j6,4,TS6)]
```

Domaines initiales des variables:

```
D(TS1)=[0..17], D(TS2)=[0..12], D(TS3)=[0..12]
D(TS4)=[0..14], D(TS5)=[0..17], D(TS6)=[0..16]
```

Exécuter ordonnancement

`precedences` : ajoute des contraintes et change les domaines

```
TS1+3 #=< TS5   TS1+3 #=< TS6   TS4+6 #=<TS3   TS5+3 #=< TS3

D(TS1)=[0..6], D(TS2)=[0..12], D(TS3)=[6..12]
D(TS4)=[0..6], D(TS5)=[3..9], D(TS6)=[3..16]
```

`machines` : ajoute des choix de contraintes, change les domaines

```
TS2+8 #=< TS1 ou TS1+3 #=< TS2   TS3+8 #=< TS1 ou TS1+3 #=< TS3...

D(TS1)=[0..0], D(TS2)=[3..4], D(TS3)=[12..12]
D(TS4)=[6..6], D(TS5)=[3..3], D(TS6)=[12..16]
```

Labeling

Dans ce cas on peut choisir la première valeur pour chaque variable

```
D(TS1)=[0..0], D(TS2)=[3..3], D(TS3)=[12..12]
D(TS4)=[6..6], D(TS5)=[3..3], D(TS6)=[12..12]
```

Solution trouvée

Contraintes réifiées

- Une contrainte réifiée `C #<=> B1` attache une variable booléenne `B1` à une contrainte simple `C`
- Si `C` est vraie alors `B=1` et si `C` est fausse alors `B=0`
- Propagation dans les deux sens

Exemples

```
ou(X,X1,X2) :-
    fd_domain([B1,B2],0,1),
    (X#=X1 #<=> B1),
    (X#=X2 #<=> B2),
    B1+B2 #>= 1.

?- fd_domain(A,1,2), fd_domain(C,3,4),fd_domain(E,2,3),ou(A,C,E).

A = 2
C = _#25(3..4)
E = 2
----
ssi(X1,V1,X2,V2) :-
    fd_domain(B,0,1), X1 #= V1 #<=> B,  X2 #= V2 #<=> B.
```