Programmation par contraintes : modélisation du problème du plan de table

UNIVERSITÉ DE NAMUR

Michel Caluwaerts

13 décembre 2017

Table des matières

1 Modélisation du problème

Le problème à modéliser est l'agencement d'une liste continue de 1 à Nb représentant la place occupée par chaque invité à un dîner. La solution est une liste de variables distinctes, et les contraintes sont portées par chaque individu pris séparément. La structure de la solution est représentée par une liste de Nb termes pers_var(n,V), liant l'individu n et ses contraintes, à V portant une solution candidate d'arrangement.

Pour faciliter la manipulation des contraintes, nous avons créé une seconde liste de Nb termes [p(Nom, Sexe, Époux, [Hobbies], [Incompatibilités],...], créée statiquement à partir de la base de données d'exemple.

2 Définition des contraintes

2.1 Alternance Homme-Femme

Nous conservons ce prédicat en première position, puisque :

— intuitivement, il réduit du facteur le plus grand l'explosion combinatoire de la solution en divisant le domaine de chaque variable en 2.

— la position de Anne étant fixée par l'énoncé, il découle que tous les hommes occuperont une position paire (y compris lorsque Nb/2 est impair, ce qu'il aurait été judicieux de comprendre tout de suite, *ndr*). Ce prédicat applique un V mod 2 sur chacun des Variables.

Auparavant, les domaines de Anne et Michel sont donc fixés à {1} et {Nb/2, Nb/2+1} .

2.2 Époux séparés

Les 3 prédicats relatifs aux époux, aux incompatibilités et aux hobbies sont construits sur le même mode : pour chaque couple pers_var, l'itération est faite sur la liste de faits *Data*, jusqu'à trouver les personnes à "éloigner" de la Variable courante.

Cet "éloignement" spécifie que

```
abs(Variable1 - Variable2) modulo Nb >= 2.
```

2.3 Liste d'incompatibilités

Idem que Époux séparés, par recherche directe des Incompatibles dans Data.

2.4 Compatibilités d'un hobby

Ici, l'optimum est de s'arrêter à une seule valeur commune entre 2 listes. Nous avons utilisé le prédicat *intersection/3*, essentiellement pour la lisibilité du code et la facilité de debugging. Sur des listes plus longues, il serait plus efficace de passer par un

```
member(X,L1), member(X,L2) ,!
```

sur 2 listes triée, avec un cut pour stopper la récursion. Ce prédicat "coûte" assez chez en instructions.

3 Comparaison de plusieurs stratégies de labeling

Nous avons testé [1] 6 méthodes de labeling (toutes autres choses restant égales par ailleurs) pour voir si une différence se dessinait, avec les résultats suivants :

```
Strategy: max
% 509,632 inferences, 0.085 CPU in 0.117 seconds (73% CPU, 6013925 Lips)
Strategy: min
% 493,356 inferences, 0.087 CPU in 0.125 seconds (70% CPU, 5656974 Lips)
Strategy: ffc
% 600,879 inferences, 0.112 CPU in 0.129 seconds (87% CPU, 5383690 Lips)
Strategy: leftmost
% 527,908 inferences, 0.085 CPU in 0.108 seconds (79% CPU, 6183910 Lips)
Strategy: up
% 527,911 inferences, 0.084 CPU in 0.109 seconds (77% CPU, 6305297 Lips)
Strategy: down
% 499,420 inferences, 0.081 CPU in 0.107 seconds (75% CPU, 6189904 Lips)
```

Pour ce faire, nous avons allongé la liste à 28 personnes, mais sans alourdir les contraintes. Comme les écarts en temps de processing ne sont pas significatifs, on pourrait en déduire que soit la taille du problème est trop réduite, soit il faudrait resserrer la liste des contraintes.

4 Impact de la longueur de la liste

La différence en temps CPU est par contre notable quand la longueur de la liste change. Toutes choses toujours égales par ailleurs, le programme sur une liste de 16 personnes prend environ 3 fois moins de temps CPU :

```
Strategy: ffc
% 120,894 inferences, 0.029 CPU in 0.063 seconds (46% CPU, 4173220 Lips)
Strategy: up
% 111,936 inferences, 0.021 CPU in 0.055 seconds (39% CPU, 5241676 Lips)
Strategy: down
% 105,968 inferences, 0.020 CPU in 0.055 seconds (37% CPU, 5302907 Lips)
Strategy: leftmost
% 111,933 inferences, 0.029 CPU in 0.063 seconds (46% CPU, 3836607 Lips)
Strategy: min
% 100,328 inferences, 0.018 CPU in 0.054 seconds (33% CPU, 5631027 Lips)
Strategy: max
% 99,897 inferences, 0.019 CPU in 0.045 seconds (43% CPU, 5189994 Lips)
```

Ce qui confirme bien qu'à niveau constant de contraintes, le nombre d'inférences augmente plus vite que la taille du problème.

1. statistiques tirées de time/1 dans la librairie *'statistics'*

5 Impact de l'ordre des prédicats

Enfin, nous avons testé le programme en sens intuitivement inverse, à savoir en ordre croissant de resserrement des contraintes :

1. Incompatibilités
2. Hobbies en commun
3. Epoux séparés
4. Alternance Homme-Femme

```
% en ordre inverse:
Strategy: down
% 512,508 inferences, 0.083 CPU in 0.110 seconds (75% CPU, 6202970 Lips)
```

Contre-intuitivement, le résultat en runtime n'est pas sensiblement modifié.

EN revanche, comme subodoré, en retirant les 2 contraintes de Hobby et Incompatibilités (qui sont pourtant peu contraignantes, puisque le dataset est assez large sur ces deux paramètres), le temps de processing se réduit sensiblement, ainsi que le nombre d'inférences.

```
%sans les hobbies:
Strategy: down
% 137,856 inferences, 0.023 CPU in 0.057 seconds (41% CPU, 6003135 Lips)

%sans les hobbies et les incompatibilités
Strategy: down
% 128,170 inferences, 0.018 CPU in 0.043 seconds (43% CPU, 6938610 Lips)
```

6 Conclusion

Nous pensons que le programme est bien fonctionnel, sur un nombre infini de listes, ce qui devrait nous permettre de tester la différence avec une configuration beaucoup plus contraignante à trouver. Enfin, dans la manière dont nous l'avons implémenté, le traitement des listes semble être plus coûteux que le labelling.