Programmation Logique et Par Contraintes Avancée Cours 8 – Optimisation

Ralf Treinen

Université Paris Cité
UFR Informatique
Institut de Recherche en Informatique Fondamentale

Université Paris Cité

treinen@irif.fr

27 février 2024

Optimisation par « Branch And Bound »

- En français : *Séparation et évaluation.*
- On lance une machine de recherche spécialisée avec deux arguments :
  1. le script,
  2. une procédure qui propage le fait que le deuxième paramètre est meilleure solution que le premier paramètre.
- La machine va chercher une solution du script qui est optimale (ici : maximale).
- Chaque fois qu'une solution X est trouvée, on relance la recherche avec la contrainte supplémentaire que la solution doit être meilleure que X.
- Fonction `SearchBest`
- Procédure `ExploreBest`

Exemple : retour à la photo de groupe

- Le nombre de préférences satisfaites fait partie de la solution (nécessaire pour comparer deux solutions).
- Ordre entre solutions : comparer le nombre de préférences satisfaites.

Exemples (photo-bb.oz) I

```
declare
proc {RevisedPhoto Root}
   Persons = [betty chris donald fred gary mary paul]
   Prefs = [betty#gary betty#mary chris#betty chris#gary
            fred#mary fred#donald paul#fred paul#donald]
   Alignment = {FD.record alignment Persons 1#{Length Persons}}
   Satisfaction = {FD.int 0#{Length Prefs}}
   fun {Satisfied P#Q}
      {FD.reified.distance Alignment.P Alignment.Q '=:' 1}
   end
in
   Root = r(satisfaction: Satisfaction
            alignment:    Alignment)
   {FD.distinct Alignment}
   {FD.sum {Map Prefs Satisfied} '=:' Satisfaction}
   Alignment.fred <: Alignment.betty   % symmetries
   {FD.distribute ff Alignment}
```

Exemples (photo-bb.oz) II

```
end

declare
proc {PhotoOrder Old New}
   % propagator for "New is better than Old"
   Old.satisfaction <: New.satisfaction
end

{ExploreBest RevisedPhoto PhotoOrder}

{Browse {SearchBest RevisedPhoto PhotoOrder}}
```


Exemples (photo-bb-wrong1.oz) I

```
declare
proc {RevisedPhoto Root}
   Persons = [betty chris donald fred gary mary paul]
   Prefs = [betty#gary betty#mary chris#betty chris#gary
            fred#mary fred#donald paul#fred paul#donald]
   Alignment = {FD.record alignment Persons 1#{Length Persons}}
   Satisfaction = {FD.int 0#{Length Prefs}}
   fun {Satisfied P#Q}
      {FD.reified.distance Alignment.P Alignment.Q '=:' 1}
   end
in
   Root = r(satisfaction: Satisfaction
            alignment:    Alignment)
   {FD.distinct Alignment}
   {FD.sum {Map Prefs Satisfied} '=:' Satisfaction}
   Alignment.fred <: Alignment.betty   % symmetries
   {FD.distribute ff Alignment}
```


Exemples (photo-bb-wrong1.oz) II

```
end

% this is wrong : syntax error (instruction expected)
declare
proc {PhotoOrder Old New}
   % propagator for "New is better than Old"
   Old.satisfaction < New.satisfaction
end

{ExploreBest RevisedPhoto PhotoOrder}

{Browse {SearchBest RevisedPhoto PhotoOrder}}
```


Exemples (photo-bb-wrong2.oz) I

```
declare
proc {RevisedPhoto Root}
   Persons = [betty chris donald fred gary mary paul]
   Prefs = [betty#gary betty#mary chris#betty chris#gary
            fred#mary fred#donald paul#fred paul#donald]
   Alignment = {FD.record alignment Persons 1#{Length Persons}}
   Satisfaction = {FD.int 0#{Length Prefs}}
   fun {Satisfied P#Q}
      {FD.reified.distance Alignment.P Alignment.Q '=:' 1}
   end
in
   Root = r(satisfaction: Satisfaction
            alignment:    Alignment)
   {FD.distinct Alignment}
   {FD.sum {Map Prefs Satisfied} '=:' Satisfaction}
   Alignment.fred <: Alignment.betty   % symmetries
   {FD.distribute ff Alignment}
```

Exemples (photo-bb-wrong2.oz) II

```
end

% this is bad : we need a better propagator here
% compare the number of nodes in the search tree !
declare
proc {PhotoOrder Old New}
   % propagator for "New is better than Old"
   if Old.satisfaction >= New.satisfaction then fail end
end

{ExploreBest RevisedPhoto PhotoOrder}

{Browse {SearchBest RevisedPhoto PhotoOrder}}
```


Using Branch and Bound

- Il est important que la “comparaison” de deux solutions est faite par un **propagateur** qui propage la contrainte que la deuxième solution est meilleure que la première.
- De cette façon on profite de la propagation pour reduire l’espace de recherche, au lieu d’énumerer bêtement toutes les possibilités.

Exemples (bb-propagation.oz) I

```
% this example shows how Branch-And-Bound profits from propagation :
% after injecting R>0, progagation yields R>=3, etc.
declare
proc {Script S}
   D R
in
   D::0#7
   R::0#20
   S=sol(d:D r:R)
   R =: 3*D
   {FD.distribute ff [D]}
end
proc {Compare Old New}
   Old.r <: New.r
end
{ExploreBest Script Compare}
```


Autres techniques d’optimisation : Recherche locale

- On part d’une solution qu’on cherche à améliorer.
- Pour toute affectation il y a une notion de voisinage, et une fonction de mesure.
- Tant que possible on passe d’un point au meilleur de ses voisins (Hill Climbing).
- Il est possible qu’on reste bloqué sur un optimum local.

Recherche locale : exemple

- Problème de n dames ($n = 5$)
- Chaque variable correpond à une ligne du damier, sa valeur sera la colonne dans laquelle se trouve la dame de cette ligne.
- On cherche à minimiser le nombre de paires de dames qui se menacent.
- Les voisins d'une affectations sont obtenues en échangeant deux lignes.

Recherche locale : exemple

1			♕		
2		♕			
3	♕				
4				♕	
5					♕

Coût : 6. Échanger lignes 2 et 3.

Recherche locale : exemple

1			♕		
2	♕				
3		♕			
4				♕	
5					♕

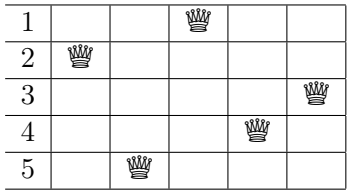
Coût : 2. Échanger lignes 2 et 5.

Recherche locale : exemple

1			♕		
2					♕
3		♕			
4				♕	
5	♕				

Coût : 0. Minimum atteint (local et global).

Recherche locale : exemple



Coût : 2. Minimum local, mais pas global.

Simulated Annealing

- Recherche locale : à chaque moment on passe à un voisin avec une certaine probabilité qui dépend :
  - de si le voisin est meilleur ou pire que l'état actuel,
  - la "température" (qui décroît avec le temps).
- En particulier il est possible d'aller vers un voisin qui est moins bien mais avec une probabilité qui décroît avec le temps.
- En plus possibilité de revenir à un état précédent avec une certaine probabilité.

Autres techniques : Simulated Annealing

- *Annealing* en français : recuit. C'est un procédé en métallurgie pour améliorer une certaine propriété (la ductilité) d'un métal, par chauffage et puis refroidissement contrôlé.
- *Simulated Annealing* : technique probabiliste pour trouver un optimum *approximatif* d'un problème difficile, inspirée du recuit de la métallurgie.
- Cette technique (ou des autres techniques probabilistes) est utile quand le problème est très complexe mais quand il est suffisant de trouver une solution qui est presque aussi bien qu'une solution théoriquement optimale.