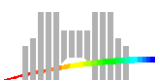
Parallélisation automatique
Calcul des dépendances

Paul Feautrier

ENS de Lyon
Paul.Feautrier@ens-lyon.fr

14 octobre 2008

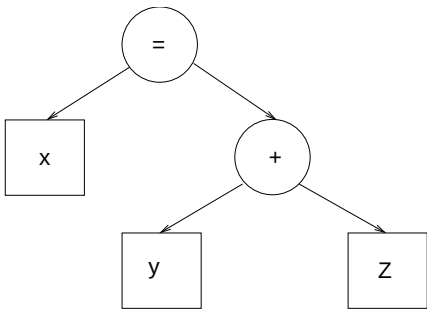


Détection des dépendances

- En principe, il faudrait calculer les dépendances entre *opérations*
- Mais il y beaucoup trop d'opérations pour que ce soit possible
- On se contente donc de représentations sommaires
- Par exemple, on dit qu'il y a dépendance entre deux *instructions* si il y a au moins *une* dépendance entre deux instances de ces instructions
- On verra plus loin des représentations plus précises.

Scalaires ; direction de dépendance

On collecte les variables lues et modifiées par visite de l'AST.
Attention cependant aux instructions complexes de C.



- On calcule ensuite les dépendances par intersection d'ensembles finis donnés en extension.
- Les dépendances sont orientées dans l'ordre d'exécution séquentielle.

Classification des dépendances.

PC	CP	PP
flow	anti	output
RAW	WAR	WAW

Problème du calcul d'adresse

- Que faire quand l'adresse d'une variable est calculée ?
  Exemples : `a[i]` ou `*p`.
- Texte identique, mais adresses différentes :

```
a[i] = ...;
i++;
a[i] : ...;
```

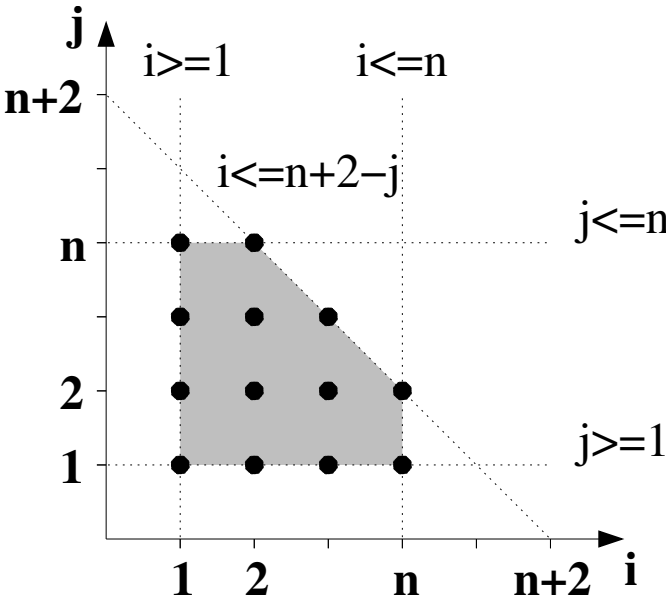
Il n'y a pas de dépendance.

- l'inverse :

```
a[i] = ...;
j = i;
a[j] : ...;
```

Il y a dépendance.

Nid de boucles




```
for(i=1 ;i<=n; i++)
  for(j=1; j<=n; j++)
    if(i+j <= 2*n-2)
      S;
```

- Contexte : une instruction est en général partie d'une structure répétitive (nid de boucles ou récursion)
- Chaque instruction engendre autant d'opérations que d'itérations
- Nommage de l'opération par son vecteur d'itération
- Le domaine d'itération est un *polyèdre*

Ordre d'exécution

- Indispensable pour orienter les dépendances.
- Dans un nid de boucle parfait, les opérations sont exécutées dans l'ordre lexicographique :

$$(i_1, \ldots, i_n) <_{\text{seq}} (j_1, \ldots, j_n) \equiv i_1 < j_1 \vee (i_1 = j_1 \ \& \ i_2 < j_2) \vee \ldots$$

- Instructions différentes : tenir compte de l'ordre textuel. Ajouter :

$$i_1 = i_2 \ \& \ \ldots \ \& \ i_n = j_n \ \& \ S <_{\text{text}} T.$$

- Nid imparfait : ne tenir compte que des boucles communes.

Interprétation des conditions de Bernstein

- Dans ce cadre, les ensembles lus et modifiés sont fonctions du vecteur d'itération. On écrit $\mathcal{R}(T, \vec{i})$, etc.
- Pour qu'une condition de Bernstein soient violée, il faut que les ensembles lues ou modifiées aient une cellule commune. Si le compilateur est correct :
  - Les tableaux ne se recouvrent pas, donc il y a un tableau commun aux deux listes, A.
  - Les bornes d'indices sont respectées, donc les indices sont égaux
- Si les deux références à A sont $A[f(\vec{i})]$ et $A[g(\vec{j})]$, il n'y a dépendance que si (équation aux indices) :

$$f(\vec{i}) = g(\vec{j}).$$

Méthodes ad hoc / I

Test de Banerjee

```
for(i=0; i<n; i++)
    a[i] = i;
```

- Deux itérations $i < j$.
- Dépendance seulement si $i = j$.
- Soit f(i) = j-i. On cherche à résoudre $f(i) = 0$ pour $i \in [0, j-1]$.
- $f(0) = j > 0$ et $f(j-1) = 1 > 0$ donc f monotone croissante et continue n'a pas de racine.

Méthodes ad hoc / II

Test du pgcd

```
for(i=0; i<n; i++)
    a[2*i] = i;
    a[2*i+1] = i+1;
```

- Dépendance seulement si $2i = 2j + 1$.
- Impossible : le pgcd des coefficients doit diviser le terme constant.

Forme générale du problème

operation domaine	$(S, \vec{i}) : A[f(\vec{i})]$ $\vec{i} \in D_S$		$(T, \vec{j}) : A[g(\vec{j})]$ $\vec{j} \in D_T$
indices		$f(\vec{i}) = g(\vec{j})$	
ordre	$i_1 < j_1$	$(S, \vec{i}) <_{\text{seq}} (T, \vec{j})$ $i_1 = j_1, i_2 < j_2$	$i_1 = j_1, i_2 = j_2$
profondeur	0	1	2

La connaissance de la profondeur de dépendance est indispensable pour l'algorithme de Allen et Kennedy, voir plus tard.

Un exemple / I

Décomposition de Cholesky

```
    for(i=1; i<=n; i++){
1:      x = a[i][i];
        for(k=1; k<i; k++)
2:          x = x - a[i][k]*a[i][k];
3:      p[i] = 1.0/sqrt(x);
        for(j=i+1; j<=n; j++){
4:          x = a[i][j];
            for(k=1; k<i; k++)
5:              x = x - a[j][k]*a[i][k];
6:          a[j][i] = x * p[i];
            }
```

Un exemple / II

- Y -a-t-il une dépendance entre 5 et 6 portant sur le tableau a ?

$(5, i, j, k) : A[j][k]]$		$(6, i', j') : A[j'][i']$
$0 \leq i \leq n$ $i+1 \leq j \leq n$ $1 \leq k \leq i-1$		$0 \leq i' \leq n$ $i'+1 \leq j' \leq n$
	$j = j', k = i'$	
$i < i'$	$i = i', j < j'$	$i = i', j = j'$
0	1	2

- Pour chaque système de contraintes, on trouve facilement une contradiction. Par exemple, à la profondeur 0 :

$$i < i' = k \leq i - 1.$$

- Existe-il une méthode systématique ?

Algorithme de Fourier-Motzkin

- On normalise toutes les contraintes sous la forme

$$h.x + k \geq 0,$$

  h vecteur de constantes, x vecteur des inconnues, k constantes.
- soit $h_1^1.x + k^1 \geq 0$ et $h_1^2.x + k^2 \geq 0$ deux contraintes telles que $h_1^1 > 0$ et $h_1^2 < 0$. On forme la nouvelle inégalité :

$$(-h_1^2 h_1^1 + h_1^1 h_1^2).x - h_1^2 k^1 + h_1^1 k^2 \geq 0.$$

- Conséquence du système initial. x_1 éliminé.
- Former toutes les combinaisons possibles.
- Eliminer toutes les inconnues. Résultat : inégalités numériques $k \geq 0$. Si l'une de ces inégalités est fausse, c'est la contradiction cherchée. Sinon, le système a des solutions.

Un exemple, III

Y a-t-il une dépendance entre 3 et 6 portant sur p ? Si oui, à quelle profondeur ?

Le test Omega

- L'algorithme de Fourier-Motzkin recherche les solutions rationnelles. Il est donc *pessimiste*
- Il existe une extension qui ne cherche que les solutions entières : le test Omega de Bill Pugh
- `www.cs.umd.edu/projects/omega`
- La complexité augmente considérablement, mais l'implémentation est extrêmement efficace.

Digression : polyèdres et polytopes

- Un polyèdre de $\mathbb{R}^n$ est l'ensemble des vecteurs à n dimensions qui satisfont à un système d'inégalités affines :

$$P = \{x \mid Ax + b \geq 0\},$$

où A est une matrice $m \times n$ et b un vecteur de dimension m.

- Un polyèdre est un objet convexe :

$$x, y \in P \Rightarrow \forall 0 \leq \lambda \leq 1 : \lambda x + (1 - \lambda)y \in P.$$

- Un polytope est un polyèdre borné.

Thèorème de Minkowski

- L'ensemble :

$$P = \{\sum \lambda_i x_i + \sum \mu_j y_j + \sum \nu_k z_k \mid \lambda_i \geq 0, \sum \lambda_i = 1, \mu_k \geq 0\}$$

  est un polyèdre
- Les x_i sont les sommets, les y_i sont les rayons et les z_i sont les lignes de P (points extrèmes de P)
- P est un polytope s'il n'a ni rayon ni ligne
- Tout polyèdre peut se mettre de façon unique sous la forme ci-dessus, que l'on note :

$$P = \mathrm{Hull}(X, Y, Z)$$

Algorithmique sur les polyèdres, I

- Si $P = \{x \mid Ax + b \geq 0\}$ et $P' = \{x \mid A'x + b' \geq 0\}$ sont deux polyèdres de même dimension, leur intersection est le polyèdre :

$$P \cap P' = \{x \mid Ax + b \geq 0, A'x + b' \geq 0\}$$

- La construction de l'intersection est triviale, mais il n'est pas évident de décider si elle est vide ou non.
- La représentation obtenue n'est pas nécessairement la plus simple.

Algorithmique sur les polyèdres, II

- L'union de deux polyèdres n'est pas nécessairement un polyèdre, parce que l'union n'est pas nécessairement convexe.
- Dans l'espace des polyèdres, la borne supérieure est la *coque convexe* de l'union (*union convexe*)
- si $P = \mathrm{Hull}(X, Y, Z)$ et si $P' = \mathrm{Hull}(X', Y', Z')$, alors

$$P \wedge P' = \mathrm{Hull}(X \cup X', Y \cup Y', Z \cup Z')$$

- La construction est triviale mais la représentation n'est pas nécessairement minimale.

Algorithmique sur les polyèdres, III

- Méthode de la double représentation : on conserve à la fois la liste des contraintes et celle des sommets, rayons et lignes.
- Pour chaque opération, on utilise la représentation la plus commode.
- Mais il faut ensuite calculer l'autre représentation.
- L'algorithme qui permet de passer de l'une à l'autre est l'algorithme de Chernikova.

Algorithmique sur les polyèdres, IV
Construction de la représentation de Minkowski

En supposant pour simplifier que le polyèdre $Ax + b \geq 0$ n'a que des sommets :

- On montre qua pour chaque sommet x il existe une sous matrice A' de A inversible et un sous vecteur b' de b tels que $A'x + b' = 0$.
- On énumère les C_m^n choix possibles, on calcule le x correxpondant par la méthode de Gauss, et on teste si $Ax + b \geq 0$.
- La complexité est donc $O(C_m^n n^3)$.

Il existe une version optimisée de cet algorithme, l'algorithme de Chernikova. Il peut être utilisé indifféremment pour passer des contraintes aux sommets ou l'inverse. Il existe une implémentation efficace de cet algorithme, la *polylib*.

Algorithmique sur les polyèdres, V
Projection

La projection de P selon la première coordonnée est définie par :

$$Q = \{y \mid \exists x_1 : x_1.y \in P\}.$$

La projection d'un polyèdre est un polyèdre.
Il existe plusieurs algorithmes de projection :

- On peut éliminer x_1 par combinaisons linéaires positives des contraintes de P (algorithme de Fourier-Motzkin).
- On peut projeter les sommets, rayons et lignes de P, puis reconstituer un système de contraintes à l'aide de l'algorithme de Cernikova.

REPRESENTATION DES DEPENDANCES

Position du problème

- Ce sont les *opérations* qui sont en dépendances
- Les tests ci-dessus permettent de savoir si deux *instructions* engendrent des opérations en dépendance
- Sauf dans les cas les plus simples, trop d'information a été perdue pour permettre l'écriture d'un programme parallèle
- Mais le graphe de dépendance détaillé est trop complexe
- On recherche des représentations synthétiques, plus ou moins approximatives
- L'approximation doit toujours être *pessimiste*

Représentation exacte

On décrit exactement les opérations en dépendances, mais sous forme d'une formule algébrique plutôt que comme une table.

Exemple Cholesky

- On cherche à carctériser la dépendance de $\langle 3, i\rangle$ vers $\langle 6, i', j'\rangle$ portant sur `p[i]`.
- Contraintes d'existence :

$$1 \leq i \leq n,\ 1 \leq i' \leq n,\ i' + 1 \leq j' \leq n$$

- Equation aux indices : $i = i'$
- Ordre d'exécution : $i < i'$ cintradictoire, et $i = i'$ redondant
- En simplifiant, on trouve qu'il y a dépendance de :

$$\{\langle 3, i\rangle \Rightarrow \langle 6, i, j'\rangle \mid 1 \leq i \leq n,\ i + 1 \leq j' \leq n\}$$

- On utilise un test normal pour vérifier que l'ensemble ainsi décrit n'est pas vide
- L'exploitation de ce genre de réprésentation est complexe, voir plus loin.

Réseau des dépendances

- On observe que le système ci-dessus comporte à la fois des équations (ordre d'exécution, équations aux indices) et des inéquations (ordre d'exécution, bornes des boucles)
- Il peut également y avoir des équations implicites :

$$a.x + b \geq 0 \ \ \& \ \ a.x + b \leq 0 \Rightarrow a.x + b = 0$$

- Si le système des équations n'a pas de solutions, la dépendance n'existe pas.

Digression : résolution d'un système linéaire en entiers

Soit :

$$Ax = b, x \in \mathbb{Z}^d$$

le système à résoudre. En général, il y a beaucoup plus d'inconnues que d'équations.
On peut mettre la matrice A sous forme de Hermite :

$$A = [HZ]U, \quad H = \begin{pmatrix} b_{11} & 0 & 0 & \cdots \\ b_{12} & b_{22} & 0 & \cdots \\ \cdots & & & \end{pmatrix}$$

où H est triangulaire inférieure à coefficients positifs, Z est une matrice nulle et où U est unimodulaire.
Il existe un algorithme polynomial pour calculer H et U.

Matrice unimodulaire

Définition Une matrice est unimodulaire si elle est entière et si son déterminant est $+1$ ou -1.

- L'inverse d'une matrice unimodulaire est unimodulaire (formules de Cramer)
- Donc on si fait le changement de variable $y = Ux, x = U^{-1}y$, y est entier si x est entier et réciproquement
- Il faut résoudre

$$[HZ]y = b$$

Solution

- Par suite de la forme spéciale de H, on voit que h_{11} doit diviser b_1 ; de même h_{22} doit diviser $b_2 - h_{12}b_1/h_{11}$, etc.
- Si une de ces conditions n'est pas remplie, le système n'a pas de solution
- Les éléments de y qui correspondent à Z peuvent prendre des valeurs arbitraires :

$$y = (c_1, \ldots, c_r, y_{r+1}, \ldots, y_d)^T \quad x = U^{-1}y.$$

- les solutions forment un *réseau*
- La méthode généralise le test du PGCD.

Polyèdre des dépendances

- On l'obtient tout simplement en ignorant la contrainte d'intégrité
- Il peut être utile de simplifier en détectant les redondances

Distance de dépendance

On suppose $N_{RS} > 0$.

- Soit i et j les vecteurs d'itération. Il est légitime de poser $d = j[1..N_{RS}] - i[1..N_{RS}]$.
- Le vecteur d satisfait un système de contraintes que l'on obtient en éliminant $i[N_{RS}..]$ et $j[N_{RS}..]$ du polyèdre des dépendances
- On peut utiliser l'algorithme de Fourier-Motzkin ou l'algorithme de Chernikova

Direction de dépendance

Le vecteur direction de dépendance a pour composantes les symboles $<, \leq, 0, \geq, >$ ou $*$.

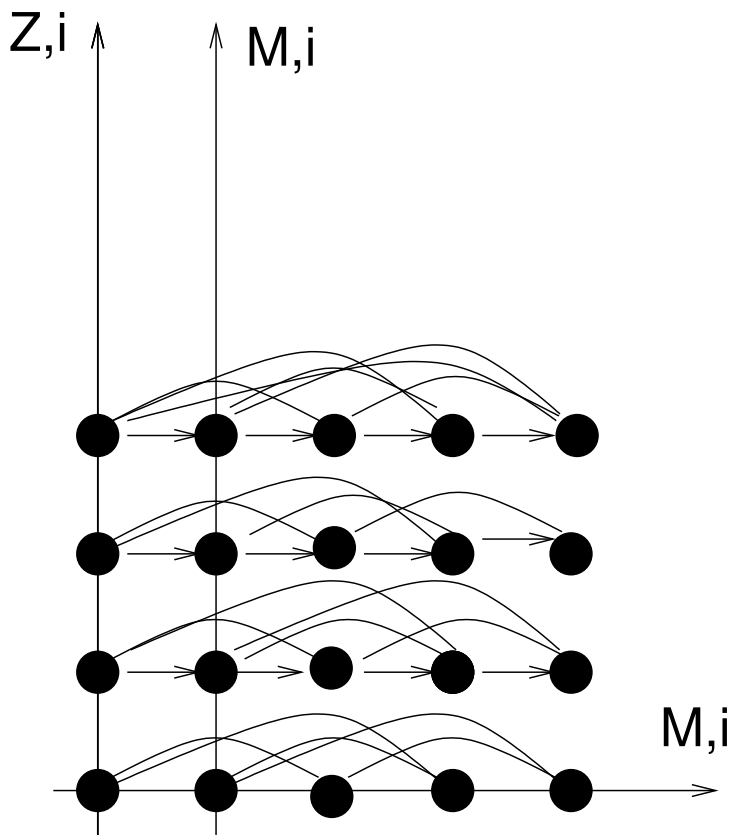
- Si la composante p est $\geq$, par example, cela signifie que $d_p \geq 0$.
- On l'obtient en étudiant le signe du vecteur de direction
- Le symbole $*$ indique que le signe n'a pas pu être déterminé.

Profondeur de dépendance

- La profondeur d'une dépendance est le nombre de zéro au début de son vecteur de direction
- Elle correspond au nombre de contraintes d'égalité retenues dans la décomposition de l'ordre lexicographique

Dépendances directes, I

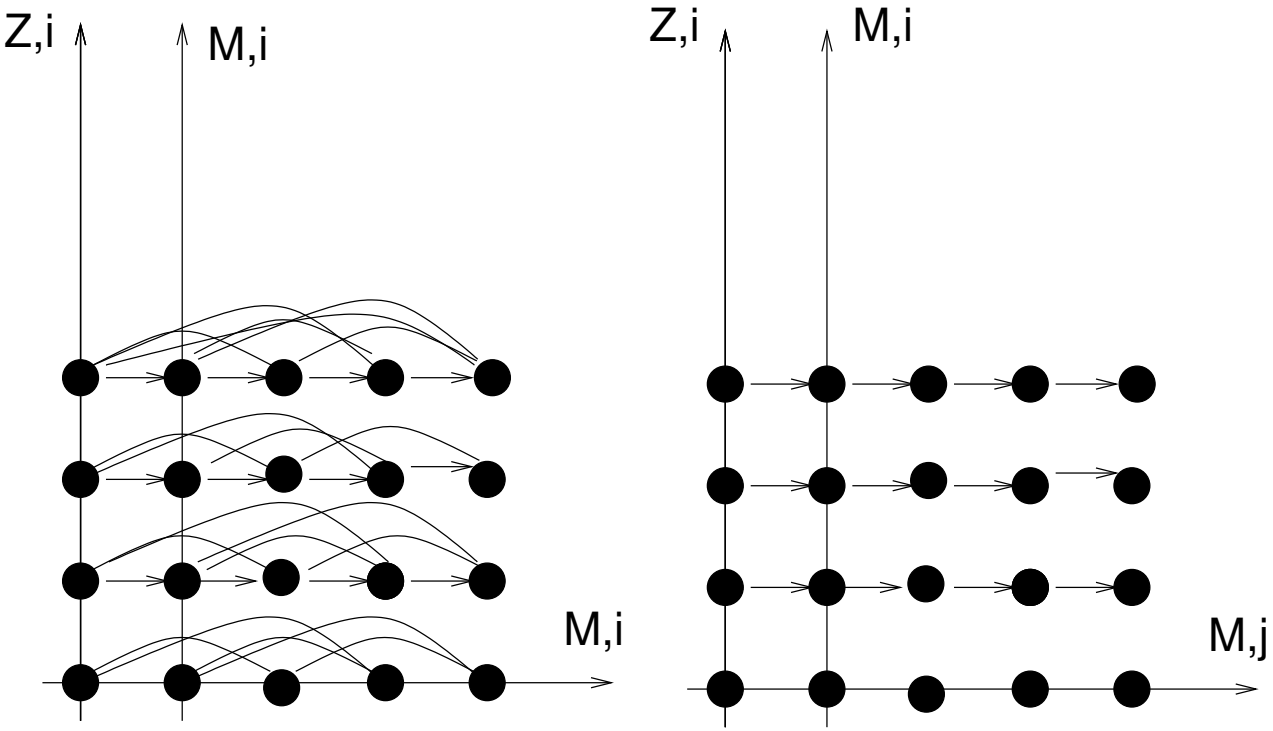
```
for(i=0; i<n; i++){
Z:  c[i] = 0.;
    for(j=0; j<n; j++)
M:      c[i] += a[i][j]*b[j];
}
```



Le graphe de dépendance détaillé a pour sommets les opérations et pour arcs les dépendances entre opérations.

Dépendances directes, II

- Le GDD est très redondant.
- Dépendances directes : ce qui reste après élimination des dépendances qui peuvent être reconstituées par transitivité.



Exemple, I

- On considère une opération de lecture, par exemple (M, i, j), qui lit $c[i]$.
- Quelle est la *source* de la valeur lue ?
- C'est la plus récente écriture dans $c[i]$ qui précède $M(i, j)$.
- Il y a deux possibilités, une instance de Z ou une instance de M.

Exemple, II

```
for(i=0; i<n; i++){
Z:   c[i] = 0.;
     for(j=0; j<n; j++)
M:       c[i] += a[i][j)*b[j];
}
```

- Cas de (Z, i').
- Ecriture dans $c[i]$: $i = i'$.
- Précède (M, i, j) : $i' \leq i$.
- Une seule solution : $i' = i$.

- Cas de (M, i', j').
- Ecriture dans $c[i]$: $i = i'$.
- Précède (M, i, j) : $i' < i \vee (i' = i \ \& \ j' < j)$.
- Le premier sous cas conduit à une contradiction. La plus tardive écriture dans le deuxième sous-cas donne $j' = j - 1$, a condition que $j \geq 1$.

Exemple, III

- Il faut maintenant trouver la plus récente solution.
- Pour $j = 0$, il n'y a qu'une possibilité, (Z, i).
- Pour $j > 0$, il y a le choix entre (Z, i) et $(M, i, j-1)$ et c'est la dernière qui est la plus récente.
- On peut résumer sous la forme :

$$\text{source}(A[i], M, i, j) = \textbf{if } j = 1 \textbf{ then } (Z, i) \textbf{ else } (M, i, j-1).$$

- Comment automatiser ?

Programmes réguliers

- Trouver la dépendance directe est un problème d'optimisation sous contraintes. On ne sait le résoudre facilement que dans le cas linéaire.
- On impose donc les contraintes suivantes :
  - Les indices des tableaux sont fonctions linéaires des compte-tours des boucles englobantes et de paramètres.
  - Les bornes des boucles sont fonctions linéaires des compte-tours des boucles englobantes et de paramètres.
  - Les paramètres ne doivent pas être modifiés dans le programme.
- Les programmes satisfaisant ces contraintes sont dits *réguliers*.
- Contrairement à ce qui se passe pour les dépendances, il est impossible d'approximer.

Construction du problème

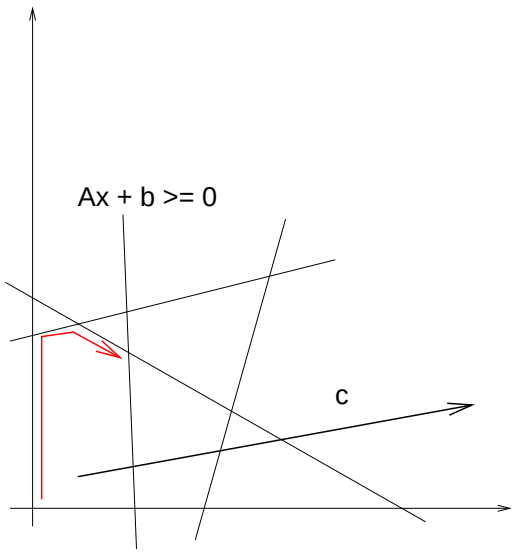
- On considère (S, i) qui lit $A[f(i)]$.
- Une source ne peut être qu'un écriture dans A. En général, il y en a plusieurs et on doit les considérer toutes. On en choisit une, (T, j) qui écrit dans $A[g(j)]$.
- Les indices doivent être égaux : $f(i) = g(j)$.
- (T, j) doit être une itération légale : $j \in D_T$.
- $(T, j) \prec (S, i)$.
- Il faut donc rechercher le maximum dans l'ordre lexicographique de l'ensemble :

$$Q(i) = \{j \mid j \in D_T, (T, j) \prec (S, i), f(i) = g(j)\},$$

qui est un polyèdre. Noter que ce polyèdre dépend de i. Le problème est paramétrique.

Programmation linéaire continue

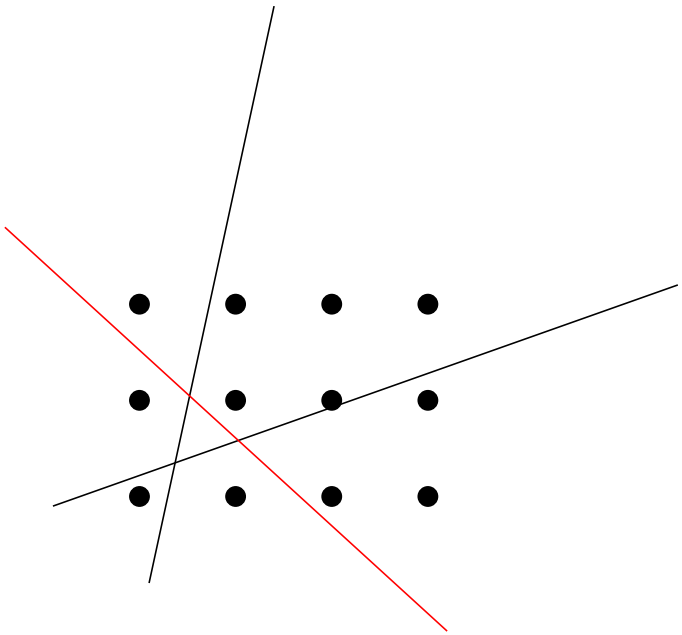
$$\begin{array}{rcl} \min & c.x & \\ Ax + b & \geq & 0 \\ x & \geq & 0 \end{array}$$



- Les contraintes définissent un polyèdre, et la solution est nécessairement en un “coin” du polyèdre.
- On va de coin en coin en suivant les arètes et en essayant de faire croitre $c.x$ le moins possible.
- On s’arrète quand on a atteint un point faisable.

Complexité : en théorie exponentielle, en pratique polynomiale.

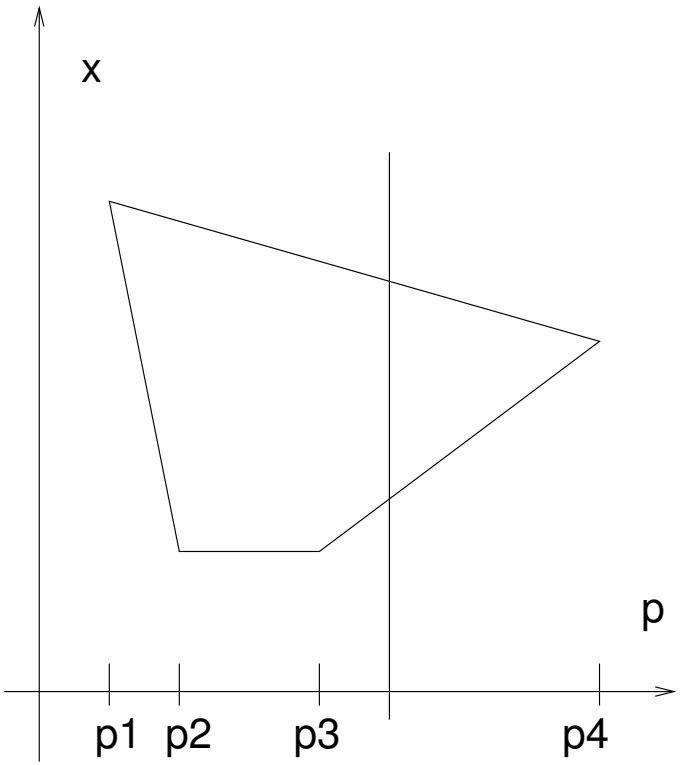
Programmation linéaire en nombres entiers (PLNE)



- On ne s'interesse qu'aux solutions entières.
- On calcule la solution rationnelle. Si elle n'est pas entière, on construit une *coupe* : une contrainte qui élimine la solution rationnelle tout en conservant la soltion entière.
- On recommence jusqu'à convergence.

Le problème devient NP-complet.

Programmation paramétrique



$$\begin{array}{rcl} \min & c.x & \\ Ax + Dp + b & \geq & 0 \\ x & \geq & 0 \end{array}$$

- p est le vecteur des paramètres (entiers).

Exercice

```
  for(k=0; k <= m+n; k++)
Z:  c[k] = 0.0;
  for(i=0; i<=m; i++)
    for(j=0; j<=n; j++)
P:    c[i+j] = c[i+j] + a[i]*b[j];
```

Quelle est la source de `c[i+j]` dans l'instruction P ?

Approximation des dépendances

- Que faire si les contraintes ne sont pas affines ?
- On peut se contenter d'un calcul approximatif *conservatif*.
- Si on dit qu'il y a une dépendance alors qu'il n'y en a pas, on perd du parallélisme : ce n'est pas grave.
- Si on dit qu'il n'y a pas de dépendance alors qu'il y en a, on engendre un programme faux : c'est grave.
- En pratique, on ignore les contraintes difficiles à traiter.
- En particulier, on ignore en général le fait que les solutions trouvées doivent être entières.