Programmation en nombres entiers

La **programmation en nombres entiers** concerne les programmes d'optimisation sous contraintes pour lesquels les variables doivent prendre des valeurs entières.

Les techniques (continues) utilisées en programmation linéaire (ou non linéaire) sont mathématiques et basées sur les notions de distance, de dérivée ou de gradient.

Elles ne peuvent généralement pas être appliquées au cas des programmes en nombres entiers (PNE), qui sont par définition discontinus.

Les méthodes d'arrondi sont généralement peu efficaces pour les PNE.

Exemple : Le problème du sac à dos consiste à choisir des objets (poids, valeur) de façon à maximiser la somme des valeurs de ces objets sans que le poids total de ces objets ne dépasse la capacité du sac à dos.

Programmation en nombres entiers

Les PNE sont aussi compliqués en linéaire qu'en non linéaire, c'est pourquoi on ne s'intéressera qu'aux **programmes linéaires en nombres entiers (PLNE)**.

Comme tout entier peut s'exprimer en ft des puissances de 2, il est possible de ramener tout PLNE à un programme en variables bivalentes ou **programme en nombre binaire (PL01)**.

On distingue 4 familles de méthodes de résolution des PLNE :

- les recherches arborescentes : procédure de séparation et évaluation (algorithme de little pour le TSP, Dakin pour la PLNE),
- les méthodes de coupes (ou troncatures) : troncatures de Gomory,
- la programmation dynamique : plus court chemin, sac à dos,
- les méthodes approchées : tabou, recuit simulé, algorithme génétique, colonie de fourmis.

Problème du voyageur de commerce

Le **problème du voyageur de commerce** (PVC ou TSP) consiste à passer par chacune des n villes, une et une seule fois, et minimiser la distance totale parcourue.

$$\begin{cases} \text{min } z = \Sigma_{i,j} d_{ij}.x_{ij} \\ \Sigma_i x_{ij} = 1, \ \forall j \\ \Sigma_j x_{ij} = 1, \ \forall i \\ \{(i,j)/x_{ij} = 1\} \text{ est un circuit} \end{cases}$$

$x_{ij} = 1$ si l'arc (i,j) de coût d_{ij} appartient au circuit hamiltonien optimal, 0 sinon.

La **matrice de dissimilarité** (ou matrice des distances) $D = (d_{ij})$ est connue pour les n villes. Elle vérifie les relations suivantes (non symétrique) :

$$\begin{cases} d(x,y) = 0 <=> x = y \\ d(x,y) \leq d(x,z) + d(z,y) \end{cases}$$

Algorithme de Little (recherche arborescente)

Matrice de dissimilarité D :

D	a	b	c	d	e	f
a	oo	1	7	3	14	2
b	3	oo	6	9	1	24
c	6	4	oo	3	7	3
d	2	3	5	oo	9	11
e	15	7	11	2	oo	4
f	20	5	13	4	18	oo

Etape 1 : un zéro par ligne

D_1	a	b	c	d	e	f
a	oo	0	6	2	13	1
b	2	oo	5	8	0	23
c	3	1	oo	0	4	0
d	0	1	3	oo	7	9
e	13	5	9	0	oo	2
f	16	1	9	0	14	oo

Etape 2 : un zéro par colonne

D_2	a	b	c	d	e	f
a	oo	0	3	2	13	1
b	2	oo	2	8	0	23
c	3	1	oo	0	4	0
d	0	1	0	oo	7	9
e	13	5	6	0	oo	2
f	16	1	6	0	14	oo

Etape 3 : matrice de pénalité

D_3	a	b	c	d	e	f
a	oo	0^2	3	2	13	1
b	2	oo	2	8	$\mathbf{0^5}$	23
c	3	1	oo	0^0	4	0^1
d	0^2	1	0^2	oo	7	9
e	13	5	6	0^2	oo	2
f	16	1	6	0^1	14	oo

Algorithme de Little

Le choix le moins pénalisant est (b, e). La longueur du chemin sera au moins de 16 = 13 (min des lignes) + 3 (min des colonnes).

Sinon, la longueur de la solution sera au moins de 16 + 5 = 21.

On supprime la ligne b et la colonne e, on interdit le choix de (e, b) et on continue avec la matrice résultante.

Etape 4 : choix (b, e)

D_4	a	b	c	d	f
a	oo	0	3	2	1
c	3	1	oo	0	0
d	0	1	0	oo	9
e	13	oo	6	0	2
f	16	1	6	0	oo

Etape 5 : matrice de pénalité

D_5	a	b	c	d	f
a	oo	0^2	3	2	1
c	3	1	oo	0^0	0^1
d	$\mathbf{0^3}$	1	0^3	oo	9
e	13	oo	6	0^2	2
f	16	1	6	0^1	oo

Le choix le moins pénalisant est (d, a) qui donnera une valeur au moins égale à 16. Sinon la solution aura pour valeur au moins 19 (pénalité de 3).

Algorithme de Little

Etape 6 : choix (d, a)

D_6	b	c	d	f
a	0^1	0^3	oo	1
c	1	oo	0^0	0^0
e	oo	3	0^2	2
f	1	3	0^1	oo

Choix de (a, c)

D_7	b	d	f
c	0^0	0^0	0^2
e	oo	0^2	2
f	0^0	0^0	oo

Choix de (c, f)

D_8	b	d
e	oo	0^0
f	0^0	0^0

La valeur du choix (d, a) nécessite de retrancher 3 à la colonne c. La solution en cours vaut donc au moins 19=16+3.

Le choix suivant (a, c) donne une solution de valeur au moins égale à 20 (19+1). Le choix de (c, f) donne une matrice qui ne contient que des 0.

La solution finale est donc : $a - c - f - b - e - d - a$ de valeur 20.

Pour terminer l'algorithme de Little, il faut revenir sur les branches non explorées : $\overline{be}(21)$, $\overline{da}(19)$, $\overline{ac}(22)$.

Algorithme de Little

La seule branche qui peut donner une solution < 20 est la branche $\overline{da}$. On repart de D_4 en interdisant le choix de (d, a). On retranche 3 à la 1ère colonne, mais on ne le rajoute pas à la valeur de la solution, car c'est déjà compté dans la pénalité de 3.

Non choix de (d, a)

D_9	a	b	c	d	f
a	oo	0^2	3	2	1
c	$\mathbf{0^{10}}$	1	oo	0^0	0^1
d	oo	1	0^4	oo	9
e	10	oo	6	0^2	2
f	13	1	6	0^1	oo

Choix de (c, a)

D_{10}	b	c	d	f
a	0^2	3	2	1
d	1	0^4	oo	9
e	oo	6	0^2	2
f	1	6	0^1	oo

La colonne f augmente de 1 la valeur minimale de la solution courante, ce qui donne 19+1=20. On n'aura pas mieux que la solution à 20.

Procédure de séparation et évaluation

Les **Procédures de Séparation et Evaluation** (PSE) permettent la résolution des problèmes NP-difficiles en énumérant implicitement des solutions de la façon suivante :

- L'arborescence des solutions est développée au cours de l'algorithme. Chaque sommet correspond à un sous-ensemble de solutions.
- **Evaluation** : pour chaque sommet S_i, on calcule une évaluation $E(S_i)$. Pour un problème de max (resp. min), la valeur doit être un majorant (resp. minorant) de l'ensemble des solutions du sommet.
- Pour un problème de max (resp. min), si l'évaluation $E(S_i)$ d'un sommet est inférieure (resp. supérieure) à la valeur d'une solution connue du problème, alors le sommet S_i ne doit pas être exploré. Il ne peut pas contenir de meilleure solution que la solution connue.
- **Séparation** : l'exploration d'un sommet S_i s'effectue en partitionnant S_i en 2 ou plusieurs sous-ensembles non vides. Les successeurs de S_i dans l'arborescence sont les sommets de ces sous-ensembles.

Procédure de séparation et évaluation

Les stratégies de parcours de l'arborescence sont :
- en profondeur d'abord,
- exploration du sommet qui a la meilleure évaluation.

L'efficacité des PSE est variable en fonction des problèmes traités. L'évaluation doit satisfaire 2 propriétés :
- rapidité : certaines fonctions d'évaluation peuvent être complexes (PL) et pénaliser la durée d'exécution de la PSE,
- justesse : écart avec la meilleure solution du sous-problème le plus petit possible.

D'autre part, il est important d'avoir une heuristique rapide (algorithme glouton) permettant d'obtenir une bonne solution du problème de manière à rapidement couper dans l'arborescence des solutions.

Problème du sac à dos

Un randonneur dispose d'un sac à dos de volume B. Il peut emporter n objets chacun de volume a_i et de valeur c_i. Le randonneur doit choisir les objets qui maximiseront la valeur totale emportée.

$$\begin{cases} max\ \Sigma_{i=1}^{n} c_i.x_i \\ \Sigma_{i=1}^{n} a_i.x_i \leq B \\ x_i \in \{0,1\} \end{cases}$$

On suppose que les objets sont indicés dans le sens des c_i/a_i décroissants.

Exemple :

$$\begin{cases} max\ z = 15x_1 + 18x_2 + 4x_3 + 7x_4 + 2x_5 + x_6 \\ 3x_1 + 4x_2 + x_3 + 3x_4 + x_5 + x_6 \leq 5 \\ x_i \in \{0,1\} \end{cases}$$

La solution du PL en variables continues ($x_i \in [0,1]$) donne une borne supérieure de la solution optimale.

Problème du sac à dos

Pour l'ensemble de toutes les solutions S_0, on obtient $E(S_0) = 24$ pour $x_1 = 1, x_2 = 1/2, x_3 = ... = x_6 = 0$.

Cet ensemble S_0 est séparé en 2 sous-ensembles :

- S_1 $(x_2 = 1)$: on trouve $E(S_1) = 23$ pour $x_1 = 1/3, x_2 = 1, x_3 = ... = x_6 = 0$,
- S_2 $(x_2 = 0)$: on trouve $E(S_2) = 21$ pour $x_1 = 1, x_2 = 0, x_3 = 1, x_4 = 1/3, x_5 = x_6 = 0$,

Le sommet S_1 qui a la meilleure évaluation, est partagé en :

- S_3 $(x_1 = 1)$: pas de solution
- S_4 $(x_1 = 0)$: on trouve $E(S_4) = 22$ pour $x_1 = 0, x_2 = 1, x_3 = 1, x_4 = ... = x_6 = 0$

Le sommet S_2 a une évaluation moins bonne que celle du sommet S_4. On a trouvé la solution entière optimale.

Métode de Dakin pour les PLNE

La méthode précédente de résolution du problème du sac à dos est généralisée à un PLNE quelconque (**methode de Dakin**).

- L'évaluation de chaque sommet est obtenue en résolvant le PL en variables continues (PLVC) associé.
- L'exploration du sommet s'arrête lorsque le PLVC n'a pas de solution ou que la solution est entière.
- Sinon, soit $x_i = v$ variable non entière dans la solution continue. L'ensemble des solutions associé au sommet est séparé en 2 sous-ensembles de la manière suivante :
  - le premier sous-ensemble de solutions est obtenu en ajoutant la contrainte $x_i \leq Int(v)$,
  - le deuxième sous-ensemble de solutions est obtenu en ajoutant la contrainte $x_i \geq Int(v) + 1$,

La séparation peut se faire sur la variable de partie fractionnaire la plus forte.

Métode de Dakin pour les PLNE

Exemple :

$$\begin{cases} \text{max } z = 3x_1 + 8x_2 \\ x_1 + 4x_2 \leq 20 \\ x_1 + 2x_2 \leq 11 \\ 3x_1 + 2x_2 \leq 22 \\ x_i \geq 0,\ x_i \text{ entier} \end{cases}$$

Le PLVC associé au sommet S_0 a pour solution $x_1 = 2$ et $x_2 = 9/2$ pour un coût de 42. L'ensemble des solutions associé au sommet S_0 est séparé en 2 selon la variable non entière $x_2 = 9/2$:

- S_1 est obtenu en ajoutant la contrainte $x_2 \geq 5$: le PLVC associé à S_1 donne $x_1 = 0$, $x_2 = 5$ pour un coût de 40.
- S_2 est obtenu en ajoutant la contrainte $x_2 \leq 4$: le PLVC associé donne $x_1 = 3$, $x_2 = 4$ pour un coût de 41.

La solution $x_1 = 3$, $x_2 = 4$ est donc optimale.

Méthode des troncatures de Gomory

La **méthode des troncatures de Gomory** est une méthode permettant de traiter des PLNE sans utiliser de recherche arborescente.

Exemple : soit le PLNE

$$\begin{cases} \text{max } z = x_1 + x_2 \\ 12x_1 - 8x_2 \leq 3 \\ 2x_2 \leq 3 \\ x_i \text{entier positif} \end{cases}$$

A l'optimum, on a :

$$\begin{cases} z + (1/12x_3 + 5/6x_4) = 11/4 \\ x_1 + (1/12x_3 + 1/3x_4) = 5/4 \\ x_2 + (1/2x_4) = 3/2 \end{cases}$$

La méthode des troncatures de Gomory est la suivante :

soit la variable de base x_i non entière, ajouter la contrainte

$$x_i + \Sigma_{x_j \in HB}(\alpha_{ij} x_j) = \beta_i \qquad \Sigma_{x_j \in HB}(f_{ij} x_j) - s_i = f_i$$

avec

- HB l'ensemble des variables hors base,
- f_{ij} partie fractionnaire de α_{ij},
- f_i partie fractionnaire de β_i.

Programmation dynamique

La **programmation dynamique** est fondée sur le **principe d'optimalité** : toute partie d'un chemin optimal est, elle-même, optimale.

Exemple : soit le plus court chemin C qui relie 2 villes x et z. Si le chemin passe par une ville y, alors la partie (x, y) est le plus court chemin entre x et y.

Applications de la programmation dynamique
- Algorithme de Ford
- Algorithme de Dijkstra
- Problème du sac à dos

Plus court chemin : Algorithme de Ford

Soit un graphe orienté et valué $G = (X, A)$ de n sommets. On numérote les sommets pour que le sommet de départ soit 0 et celui d'arrivée $n - 1$. Chaque arc (i, j) est de valeur $v(i, j)$.

On associe au chemin $C =< (i_1, i_2), (i_2, i_3), ... >$ une longueur égale à la somme des valeurs des arcs qui constituent le chemin. Le plus court chemin du sommet 0 au sommet $n - 1$ est le chemin de longueur minimum qui va de 0 à $n - 1$.

L'algorithme de Ford est le suivant (en $O(2^n)$) :

1. Affecter pour tout sommet i, une valeur λ_i infinie, et faire : $\lambda_0 = 0$.
2. Pour tout sommet j tel que $\lambda_j > \lambda_i + v(i, j)$, faire $\lambda_j = \lambda_i + v(i, j)$.
3. Retourner en 3 tant que des λ_i sont modifiés.

Plus court chemin : Algorithme de Dijkstra

Soit un graphe orienté et valué $G = (X, A)$ de n sommets. On numérote les sommets pour que le sommet de départ soit 0 et celui d'arrivée $n - 1$. Chaque arc (i, j) est de valeur $v(i, j)$.

L'algorithme de Dijkstra est le suivant (en $O(m.log(n))$) :

1. $\forall i \neq s, \lambda_i = +oo$; $\lambda_0 = 0$; tout sommet est non traité
2. Tant que tous les sommets ne sont pas traités faire
3. soit i un sommet non traité, de λ_i minimal
4. pour tout arc (i, j), faire
5. si $\lambda_i + v(i, j) < \lambda_j$, alors $\lambda_j = \lambda_i + v(i, j)$
6. le sommet i est traité

Flot de valeur maximale

On appelle **réseau de transport** un graphe orienté sans boucle comportant une **source** x_1 et un **puit** x_p. Depuis x_1, il existe un chemin vers tout autre sommet x_k et de tout sommet x_k il existe un chemin vers x_p. Tout arc u est valué par un entier positif $c(u)$ nommé **capacité** de l'arc u.

Le problème à résoudre consiste à acheminer une quantité maximale de x_1 vers x_p en tenant compte des capacités.

En tout sommet x, on a une loi de conservation : la somme des flux arrivant sur x est égale à la somme des flux partant de x.

Définition 6. *une* **chaîne améliorante** *est une chaîne élémentaire* $\mu = (x_1, ..., x_p)$ *telle que :*

- *aucun arc direct ne soit saturé (flux associé strictement inférieur à la capacité)*
- *les flux des arcs indirects soient strictement positifs*

Flot de valeur maximale : Algorithme de Ford-Fulkerson

1. Tant qu'il existe μ, une chaîne améliorante faire
2. augmenter le flux sur μ de la façon suivante :
   - calculer $\delta^+ = min\{c_u - \phi_u\}$, où u est un arc direct de G,
   - calculer $\delta^- = min\{\phi_u\}$, où u est un arc indirect de G,
   - $\delta = min\{\delta^+, \delta^-\}$
   - pour tout arc direct u faire : $\phi_u = \phi_u + \delta$
   - pour tout arc indirect u faire : $\phi_u = \phi_u - \delta$

Flot de valeur maximale : Algorithme de Ford-Fulkerson

Procédure de marquage pour déterminer une chaîne améliorante

1. Initialement, la source O est marquée et les autres sommets sont non marqués.
2. Tant que possible, choisir un sommet x non marqué vérifiant l'une des définitions 3 ou 4 suivantes :
3. x est extrémité d'un arc (y, x) tel que y est marqué et $\phi_{(x,y)} < c_{(x,y)}$; marquer $+$ le sommet x
4. x est origine d'un arc (x, y) tel que y est marqué et $\phi_{(x,y)} > 0$; marquer - le sommet x
5. $p(x) = y$ (le sommet y a permis de marquer le sommet x : le prédécesseur de x dans le marquage, soit $p(x)$, est le sommet y).

Problème du sac à dos

$$\begin{cases} max\ \Sigma_{i=1}^{n} c_i.x_i \\ \Sigma_{i=1}^{n} a_i.x_i \leq B \\ x_i \text{ entier positif ou nul} \end{cases}$$

Programmation dynamique :

A l'itération k, on détermine une solution optimale correspondant à un sac à dos de capacité k. Si $z(k)$ est la valeur de la solution optimale, on a :

$z(k) = max_{j/a_j \leq k}\{c_j + z(k - a_j)\}$

avec comme C.I. : $z(k) = 0$ si $k < min_j\{a_j\}$.

En pratique, on construit un tableau à 3 colonnes k, $z(k)$ et $j(k)$. $j(k)$ contient les indices j tels que $z(k) = c_j + z(k - a_j)$.

Programme de transport

Un programme de transport a pour but d'acheminer des marchandises au moindre coût depuis m origines vers n destinations. On suppose connu :

- la quantité disponible a_i de l'origine i,
- la quantité demandée b_j de la destination j,
- la matrice des coûts unitaires $C = (c_{ij})$.

Le problème est de minimiser la somme des coûts de transport :

$$\begin{cases} \text{min } z = \Sigma_{ij}\ c_{ij}.x_{ij} \\ \Sigma_{j=1}^{n}\ x_{ij} = a_i,\ (i = 1, 2, ..., m) \\ \Sigma_{i=1}^{m}\ x_{ij} = b_j,\ (j = 1, 2, ..., n) \\ x_{ij} \text{ entier positif ou nul} \end{cases}$$

Méthode :

1. Construire une solution initiale par la méthode du **coin Nord-Ouest**,
2. Améliorer la solution courante par la méthode du **stepping-stone**.

Méthode du coin Nord-Ouest

- On transporte le maximum de I vers 1 = min(9,18)
- Si l'origine est vide, on passe à l'origine suivante (même colonne),
- Si la destination est satisfaite, on passe à la destination suivante (même ligne),

Exemple :

i - j	1	2	3	4	5	6	a_i
I	9	9					18
II		2	28	2			32
III				4	10		14
IV					4	5	9
b_j	9	11	28	6	14	5	73

Méthode du steeping-stone

- calcul d'un potentiel U_i pour chaque ligne i,
- calcul d'un potentiel V_j pour chaque colonne j,
- calcul des coûts marginaux : $\delta_{i,j} = U_i + c_{i,j} - V_j$.
- pour chaque coût marginal négatif, rechercher le cycle de substitution permettant de réaliser le transport correspondant, la quantité maximale déplaçable et le gain correspondant,
- choisir la transformation pour laquelle le gain est maximum.

i - j	1	2	3	4	5	6	a_i
I	12	27	61	49	83	35	18
II	23	39	78	28	65	42	32
III	67	56	92	24	53	54	14
IV	71	43	91	67	40	49	9
b_j	9	11	28	6	14	5	73

Valeur solution coin Nord-Ouest = 3700

Méthode du steeping-stone

Calcul des potentiels U_i et V_j :

i - j	1	2	3	4	5	6	U_i
I	9/12	9/27	61	49	83	35	39-27
II	23	2/39	28/78	2/28	65	42	0
III	67	56	92	4/24	10/53	54	28-24
IV	71	43	91	67	4/40	5/49	57-40
V_j	12+12	0+39	0+78	0+28	4+53	17+49	

Calcul des coûts marginaux $\delta_{i,j} = U_i + c_{i,j} - V_j$:

i - j	1	2	3	4	5	6	U_i
I	9/12	9/27	-5	33	38	-19	12
II	-1	2/39	28/78	2/28	8	-24	0
III	47	21	18	4/24	10/53	-8	4
IV	64	21	30	56	4/40	5/49	17
V_j	24	39	78	28	57	66	

Améliorations possibles : II,1($-1 \times 2 = -2$) ; I,3($-5 \times 9 = -45$) ; III,6($-8 \times 5 = 40$) ; II,6($-24 \times 2 = -48$) ; I,6($-19 \times 2 = -38$).

Méthode du steeping-stone

Choix de I,3 et II,6 (gain de 45+48=93, nouvelle solution à 3607)

Calcul des potentiels U_i et V_j :

i - j	1	2	3	4	5	6	U_i
I	9/12	27	9/61	49	83	35	98-61
II	23	11/39	19/78	28	65	2/42	62-42
III	67	56	92	6/24	8/53	54	0
IV	71	43	91	67	6/40	3/49	53-40
V_j	37+12	20+39	20+78	24	53	13+49	

Calcul des coûts marginaux $\delta_{i,j} = U_i + c_{i,j} - V_j$:

i - j	1	2	3	4	5	6	U_i
I	9/12	5	9/61	62	67	10	37
II	-6	11/39	19/78	24	32	2/42	20
III	18	-3	-6	6/24	8/53	-8	0
IV	35	-3	6	56	6/40	3/49	13
V_j	49	59	98	24	53	62	

Méthode du steeping-stone

Améliorations possibles : II,1($-6 \times 9 = 54$) ; III,2($-3 \times 3 = -9$) ; III,3($-6 \times 3 = 18$) ; III,6($-8 \times 3 = -24$) ; IV,2($-3 \times 2 = -6$).

Choix de II,1 et III,6 (gain de 54+24=78, solution optimale à 3529)

i - j	1	2	3	4	5	6	a_i
I			18/61				18
II	9/23	11/39	10/78			2/42	32
III				6/24	5/53	3/54	14
IV					9/40		9
b_j	9	11	28	6	14	5	73

Problème d'affectation

Exemple : 5 personnes A, B, C, D et E doivent être affecter à 5 postes a, b, c, d, e. Chaque personne a classé les postes par ordre de préférence.

	a	b	c	d	e
A	1	2	3	4	5
B	1	4	2	5	3
C	3	2	1	5	4
D	1	2	3	5	4
E	2	1	4	3	5

=>

	a	b	c	d	e
A	0	1	2	1	2
B	0	3	1	2	0
C	2	1	0	2	1
D	0	1	2	2	1
E	1	0	3	0	2

On soustrait le minimum de chaque ligne et le minimum de chaque colonne.

Si on affecte le zéro unique de la ligne A ; B sera affecté à e ; C à c ; aucun 0 ne peut être affecter à D et E peut être affecté à b ou d. Il n'est pas possible d'obtenir une solution du problème en choisissant que des 0.

Problème d'affectation : méthode hongroise

Les zéros supprimés par suite d'une affectation d'un autre zéro sur la même ligne ou colonne seront appelés les **zeros barrés**. Les zéros affectés seront appelés les **zéros encadrés**.

La méthode consiste à :

1. marquer toute ligne n'ayant pas de zéro,
2. marquer toute colonne ayant un zéro barré sur une ligne marquée,
3. marquer toute ligne ayant un zéro encadré dans une colonne marquée et revenir en 2 jusqu'à ce que le marquage ne soit plus possible,
4. rayer les lignes non marquées et les colonnes marquées,
5. retrancher le plus petit élément du sous-tableau restant à tous les éléments non rayés et ajouter le aux éléments rayés 2 fois.

Problème d'affectation : méthode hongroise

La matrice initiale est à gauche. On marque la ligne D (1), puis la colonne a (2), puis la ligne A (3). Après avoir effectué les points 4 et 5, on obtient la solution de droite :

	a	b	c	d	e
A	$\boxed{0}$	1	2	1	2
B	Ø	3	1	2	$\boxed{0}$
C	2	1	$\boxed{0}$	2	1
D	Ø	1	2	2	1
E	1	$\boxed{0}$	3	Ø	2

	a	b	c	d	e
A	$\boxed{0}$	0	1	0	1
B	1	3	1	2	$\boxed{0}$
C	3	1	$\boxed{0}$	2	1
D	0	$\boxed{0}$	1	1	0
E	2	0	3	$\boxed{0}$	2

Colonie de Fourmis

Les algorithmes de fourmi ont été proposés par Dorigo dans les années 90 en tant qu'approche multi-agent pour résoudre des problèmes combinatoires comme le TSP. Ces algorithmes ont été inspirés des colonies de fourmis réelles.

- Les fourmis sont des insectes sociaux.
- Comportement orienté vers la survie de la colonie.
- Haut degré de structuration de la colonie (simplicité de l'individu).
- Capacité à récolter de la nourriture et à trouver le plus court chemin entre le nid et la source de nourriture (Emergence).
- Dépôt de phéromone permettant de se repérer et d'indiquer à ses congénères le chemin vers la source de nourriture.
- Expérience du double pont qui montre que les fourmis choisissent un chemin suivant une loi aléatoire proportionnelle aux taux de phéromone de ce chemin.

Optimisation par Colonie de Fourmis

Similitudes avec les fourmis réelles :
- colonie de fourmis artificielles (une fourmi artificielle est capable de construire une solution)
- utilisation de phéromones pour la communication
- prise de décision stochastique à partir des données du problème et des informations locales

Différences avec les fourmis réelles :
- mémorisation des états précédents
- dépôt de phéromone fonction de la qualité de la solution
- le dépôt de phéromone peut avoir lieu après la construction de la solution
- procédures particulières (optimisation locale, retour-arrière, exploration) pour améliorer la recherche

Optimisation par Colonie de Fourmis : TSP

- L'algorithme exécute t_{max} itérations : à chaque itération, m fourmis construisent un cycle en se déplaçant d'une ville à une autre.
- Chaque fourmi contient une liste tabou des villes déjà visitées.
- A la ville i, on calcule un poids $a_{ij}(t)$ d'aller vers j :

$$\forall\, j \;\in\; N_i,\; a_{ij}(t) = \frac{\tau_{ij}(t)^{\alpha} \times \eta_{ij}^{\beta}}{\sum_{l \in N_i} (\tau_{il}(t)^{\alpha} \times \eta_{il}^{\beta})}$$

  avec $\tau_{ij}(t)$ le taux de phéromone, $\eta_{ij} = 1/d_{ij}$ et N_i l'ensemble des voisins du noeud i.
- La probabilité pour la fourmi k d'aller vers la ville j depuis la ville i est donnée par :

$$p_{ij}^{k}(t) = \frac{a_{ij}(t)}{\sum_{l \in N_i^k} a_{il}(t)}$$

  avec N_i^k l'ensemble des voisins du noeud i non visités par la fourmi k.

Optimisation par Colonie de Fourmis : TSP

- Quand les fourmis ont terminé leur cycle, l'évaporation des phéromones a lieu et chaque fourmi k dépose une quantité de phéromone $\Delta_{ij}^k(t) = 1/L^k(t)$ si (i, j) appartient au cycle de la fourmi, 0 sinon.
- En pratique cela donne : $\tau_{ij}(t) = (1 - \rho)\tau_{ij}(t) + \sum_k \Delta_{ij}^k(t)$
- Au départ, on initialise les valeurs suivantes :
  - $\tau_{ij}(0)$: valeur initiale de phéromone sur tous les arcs (petite constante positive).
  - Le nombre de fourmi $m = n$ (nombre de villes).
  - $\alpha = 1$ (phéromone) ; $\beta = 5$ (distance) ; $\rho = 0.5$ (évaporation)
- Introduction également de fourmis élitistes : les arcs utilisés par une fourmi qui a généré le meilleur tour récupèrent un dépôt de phéromone supplémentaire.

Optimisation par Colonie de Fourmis

Autres problèmes traités par les colonies de fourmis :

- Problème d'affectation quadratique : affecter au moindre coût n entreprises à n localisations (généralisation du TSP).
- Problème d'ordonnancement (Job-Shop) : affectation de J tâches à M machines de façon à minimiser la durée totale d'exécution.
- Problème de routage de véhicules : on dispose d'un entrepôt et de M véhicules de capacité D pour servir des clients, caractérisés par une demande d_i et un temps de service δ_i. Il faut servir tous les clients une seule fois et minimiser le coût total.
- ...

Problèmes dynamiques de routage dans les réseaux informatiques / de télécommunication :

- routage dans les réseaux orientés connexion : tous les paquets d'une même session suivent le même chemin sélectionné durant une phase préliminaire.
- routage dans les réseaux sans connexion : les paquets de données d'une même session peuvent suivre différents chemins.