TP d'UMIN215

Année 2011-12

Version 1.3

Université de Montpellier
Place Eugène Bataillon
34095 Montpellier Cedex 5

Rodolphe Giroudeau
161, rue Ada
34392 MONTPELLIER Cedex 5
TEL : 04-67-41-85-40
Mail : rgirou@lirmm.fr

Méthode de Résolution de problèmes $\mathcal{NP}$-complet
TD – Séance n° 1

---

1 Partie théorique

1.1 Programmation linéaire en nombres entiers

Exercice 1 – Sur le problème de la couverture sommet minimale : trois approches différentes

Nous considérons le problème suivant :

Instance : Soit un graphe $G = (V, E)$.

Question : Trouver $S \subseteq V$ tel que :

– Toutes les arêtes de G sont incidentes au moins à un sommet de S avec un coût minimum

1. Utilisation de la programmation linéaire en nombres entiers.
   Nous pouvons modéliser ce problème par un programme linéaire en nombres entiers :

$$\begin{cases} \min z = \sum_{j=1}^{n} x_i \\ x_r + x_s \geq 1, \forall \{v_r, v_s\} \in E \text{ et} \\ x_j \in \{0,1\} j = 1 \ldots n \end{cases}$$

Algorithm 1 Algorithme approché pour le problème Vertex cover

---

Exprimer l'instance du problème par un programme linéaire en nombres entiers.
Résoudre le programme relaxé. Soit $X = (x_1, x_2, \ldots, x_n \in \mathbb{R}^{\geq 0})^{n}$
Soit $S_X = \{v_i | x_i \geq 1/2\}$

   (a) Justifier le programme linéaire en nombres entiers précédent.
   (b) Pourquoi ne peut-on avoir $x_s + x_r = 1$?
   (c) Montrer que le programme linéaire en nombres entiers est une borne inférieure de toute solution optimale.
   (d) Montrer que la relaxation des contraintes d'intégrité implique $x_r \geq 1/2$ ou $x_s \geq 1/2$.
   (e) Montrer que l'algorithme 1 conduit à un algorithme approché avec une performance relative de deux.
   (f) Nous allons étudier une version étendu du problème précedent dans lequel le graphe est valué.

      i. Donner le programme linéaire dans le cas où les arêtes $(i, j) \in E$ possèdent un poids w_{ij} c'est à dire que nous considérons le problème suivant :
      Instance : Soit un graphe $G = (V, E)$. et une fonction $w : V \times V \rightarrow \mathbb{N} - \{0\}$
      Question : Trouver $S \subseteq V$ tel que :
      – Toutes les arêtes de G sont incidentes au moins à un sommet de S avec un coût minimum $(\sum_{i,j \in S} w_{ij})$.
      ii. Donner le programme linéaire en nombres entiers en utilisant une matrice A. Quelle est la caractéristique de la matrice A ?
      iii. Montrer qu'il existe un algorithme 2-approché pour le problème pondéré.

2. Utilisation d'un algorithme basé sur la recherche d'un couplage maximal. Dans la suite nous allons proposer un algorithme approché pour le problème, sans pondération sur les arêtes, en utilisant un simple algorithme basé sur la recherche un couplage maximal.
   (a) Nous considérons l'algorithme suivant :

Algorithm 2 Algorithme donnant la couverture sommet approchées en utilisant la notion de couplage

---

$C \Leftarrow \emptyset$
$E' \Leftarrow E(G)$
while $E' \neq \emptyset$ **do**
  soit (u, v) une arête arbitraire de E'
  $C \leftarrow C \bigcup \{u, v\}$
  supprimer de E' toutes les arêtes incidentes soit à u soit à v.
end while
Retourner C

   Montrer que l'algorithme est un algorithme approché de performance relative de 2.
   (b) Trouver un graphe pour lequel la borne de deux est atteinte.
   (c) Une amélioration possible consisterait à chaque pas de l'algorithme 2 de prendre un sommet de plus haut degré afin de supprimer à chaque étape de l'algorithme le maximum d'arêtes. Ensuite, on supprime toutes ses arêtes incidentes. Voici l'algorithme,

---

Algorithm 3 Algorithme donnant la couverture sommets approchées en utilisant les sommets de plus haut degré

$C \Leftarrow \emptyset$
$E' \Leftarrow E(G)$
while $E' \neq \emptyset$ **do**
  soit u un sommet de plus haut degré
  $C \leftarrow C \bigcup \{u\}$
  supprimer de E' toutes les arêtes incidentes à u.
end while
Retourner C

   Appliquer l'algorithme 3 sur le graphe donné par la figure 1. Conclusion ?

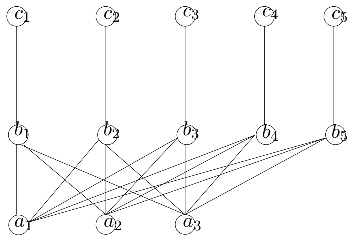

FIG. 1 – Pire des cas

3. Utilisation d'un algorithme primal-dual

Exercice 2 – Sur le problème de la couverture d'ensembles

Nous considérons le problème de la couverture d'ensembles :

Soit $E = \{e_1, \ldots, e_n\}$, des sous-ensembles de E noté $S_1, S_2, \ldots, S_m$ avec $S_j \subseteq E$, et un poids $w_j \geq 0$ pour chaque sous-ensemble S_j. Le but est de trouver une collection de sous-ensembles de poids minimum qui couvre tous les éléments de E, c'est-à-dire nous voulons trouver $I \subseteq \{1, \ldots, m\}$ qui minimise $\sum_{j \in I} w_j$ tel que $\cup_{j \in I} S_j = E$. Si $w_j = 1, \forall j$ le problème est sans poids.

1. Modéliser ce problème par un programme linéaire en nombres entiers.
2. Donner une procédure d'arrondis sur les variables en fonction de $f = \max_{i=1,\ldots,n} f_i$ avec $f_i = |\{j : e_i \in S_j\}|$.
3. Montrer que la procédure d'arrondie précedente garantie une solution réalisable.
4. Montrer l'existence d'un algorithme f-approché.
5. Que retrouve-t'-on quand $f_i = 2$?

Exercice 3 – Sur le problème du couplage maximum de poids minimum

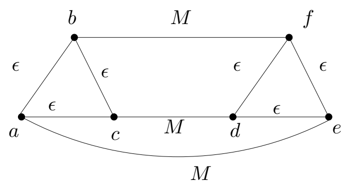

FIG. 2 – Cas pathologique

1. Donner le programme linéaire en nombres entiers qui modélise ce problème.
2. Donner les équations des contraintes pour le graphe donné par la figure 2.
3. Donner une solution optimale notée $z(ILP)$.
4. Donner une solutuion pour le programme relaxé noté $z(LP)$.
5. Conclure sur la pertinence de la formulation.

1.2 Problèmes appartenant à la classe APX

Exercice 4 – Sur le problème de la coupe maximum

Instance : Soit $G = (V, E)$ un graphe.

Question : Trouver $S \subset V$ tel que $(S, V - S)$ soit maximum.

Algorithm 4 Algorithme approché pour le problème max-cut

$S = \varnothing$
while Il existe un sommet $v \in V$ tel que le déplacement de v d'un coté de la coupe $(S, V - S)$ à l'autre augmente la valeur de celle-ci) **do**
  Prendre un sommet u qui augmente la valeur de la coupe si nous le déplaçons, et le déplacer
end while
Renvoyer $(S, V - S)$

1. Donner la complexité de l'algorithme 4
2. Montrer que l'algorithme 4 produit un algorithme deux approché. (Considérons (Y_1, Y_2) la coupe donnée par l'algorithme. Montrer que chaque sommet dans Y_1 admet au moins autant d'arêtes dans Y_2 que de d'arêtes dans Y_1.)
3. Contruisez une instance pour laquelle la borne de deux est atteinte.

1.3 Constructions de PTAS

Sur le problème de la partition :

Instance : étant donné un ensemble A de n objets $A = \{a_i\}$ $(1 \leq i \leq n)$ de poids entiers $p(a_1),\ p(a_2), \ldots,\ p(a_n)$,

Question : Est-il possible de les partager en deux sous-ensembles de même poids total P ?

Exercice 5 – Sur le problème de Partition

1. Montrer que pour $r \geq 2$, la solution $A, \varnothing$ est une r-approximation.
2. Nous supposons maintenant que $r < 2$. Nous posons $w(A) = \sum_{a_i \in A} p(a_i)$. De même, $w(Y_i) = \sum_{a_i \in Y_i} p(a_i)$, pour $i = 1, 2$, et $L = w(A)/2$. Nous supposons également que $w(Y_1) \geq w(Y_2)$, et que a_h est le dernier objet introduit dans Y_1 (voir figure 3).
   (a) Montrer $w(Y_1) - L \leq p(a_h)/2$.
   (b) Que se passe-t'il si a_h est inséré dans Y_1 durant la **première phase** ?
   (c) Supposons maintenant que a_h soit inséré durant la **seconde phase**. Comparer $p(a_h)$ à $p(a_j)$, $1 \leq j \leq k(r)$. Montrer que $2L \geq p(a_h)(k(r) + 1)$.
   (d) Donner une borne inférieure pour toute solution optimale en fonction de L.
   (e) Montrer que le ratio est majoré par r. Conclure.
3. Donner la complexité de l'algorithme.

Exercice 6 – Sur le problème du sac à dos simple

Instance : Soit $w_1, w_2, \ldots, w_n$ et b des nombres. Soit $T \subseteq \{1, \ldots, n\}$ (avec $cost(T) = \sum_{i \in T} w_i$)

Question : Trouver T tel que $cost(T)$ soit maximum ($cost(T) \leq b$).

Algorithm 5 Construction d'un PTAS pour le problème de la pPartition

if $r \geq 2$ **then**
  retourner $A, 0$
else
  Trier par ordre décroissant en fonction des poids
  Soit $(a_1, \ldots, a_n)$
  $k(r) := \lceil \frac{2-r}{r-1} \rceil$
  Première phase
  Trouver une partition optimale Y_1, Y_2 de $a_1, \ldots, a_{k(r)}$
  Seconde phase
  for $j := k(r) + 1$ à n **do**
    if $\sum_{x_1 \in Y_1} p(a_i) \leq \sum_{x_i \in Y_2} p(a_i)$ **then**
      $Y_1 := Y_1 \cup \{a_j\}$
    else
      $Y_2 := Y_2 \cup \{a_j\}$
    end if
  end for
  Retourner Y_1, Y_2
end if

1. Construction d'un algorithme approché :

Algorithm 6 Algorithme glouton

Trier les entiers $w_1, \ldots, w_n$. Nous supposerons par la suite que $b \geq w_1 \geq w_2 \geq \ldots \geq w_n$
$T := \varnothing$; $cost(T) = 0$;
for $i = 1$ à n **do**
  if $cost(T) + w_i \leq b$ **then**
    $T := T \cup \{i\}$; $cost(T) := cost(T) + w_i$
  end if
end for

   (a) Donner la complexité de l'algorithme 1.
   (b) Montrer quil existe un indice (noté par la suite $j+1$) pour lequel $cost(T) + w_{j+1} > b$. En déduire que $cost(T) \geq b/2$. Etudier le cas où $j = 1$ et $j \geq 2$.
   (c) En conclure que l'algorithme 1 admet une performance relative de deux.

2. Construction d'un schéma d'approximation :
   (a) Montrer que l'algorithme 2 admet ue complexité de $O(n^{k+1})$. Pour cela calculer d'une part le coût de création des ensembles S et d'autre part le coût du passage de S à S^*.
   (b) Soit $M = \{i_1, i_2, \ldots, i_p\}$ avec $i_1 < i_2 < \ldots < i_p$ l'ensemble des indices d'une solution optimale pour l'instance I i.e. $cost(M) = OPt(I)$.
      i. Que peut-on dire si $p \leq k$ ¿
      ii. Nous supposons maintenant que $k > p$. Nous considérons l'ensemble des indices $P = \{i_1, i_2, \ldots, i_k\}$ associé à l'ensemble S. Les objets $w_{i_1}, \ldots, w_{i_k}$ sont les k objets ayant un poids maximum. Soit P^* l'ensemble des indices de S^*.

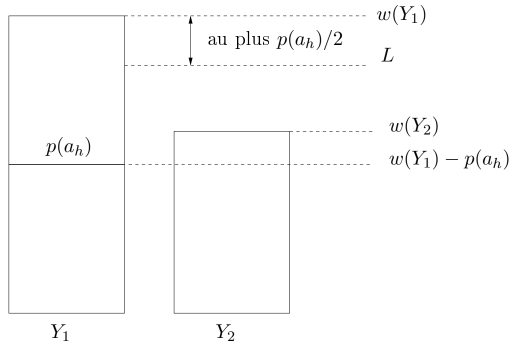

FIG. 3 – Analyse de l'algorithme 1.3

Algorithm 7 Construction d'un PTAS pour le problème du sac à dos simple

Trier les entiers $w_1, \ldots, w_n$. Nous supposerons par la suite que $b \geq w_1 \geq w_2 \geq \ldots \geq w_n$
$k := \lceil \frac{1}{\epsilon} \rceil$
Pour chaque ensemble $S \subseteq \{1, 2, \ldots, n\}$ avec $|S| \leq k$ et $\sum_{i \in S} w_i \leq b$, étendons S à S^* en utilisant un algorithme glouton (celui présenté dans l'algorithme 1). Les ensemble S sont crées de manière séquentielle dans l'ordre lexicographique jusqu' à que le meilleur S^* soit trouvé c'est à dire un ensemble S^* tel que $cost(S^*)$ soit maximal.

         A. Si $P^* = M$, conclure.
         B. Supposons que $P^* \neq M$. Montrer que 'il existe $i_q > i_k \geq k$ tel que $cost(P^*) + w_{i_q} > b \geq cost(M)$.
         C. Montrer que $w_{i_q} \leq \frac{cost(M)}{k+1}$.
         D. Evaluer le ratio $R(I, \epsilon) = \frac{cost(M)}{cost(P^*)}$ et conclure

1.4 Utilisation de méthodes exactes

1.4.1 Programmation dynamique

Résoudre un problème d'optimisation par la programmation dynamique n'est pas touours facile. La méthode, qui est une méthode exacte, consiste en effet à plonger le problème proposé dans un problème plus général, dépendant de paramètres entiers, le problème initial correspondant à une valeur précise de ceux-ci. On essaie alors de résoudre le problème par itération sur ces paramètres entiers.

Les problèmes auxquels s'adresse cette méthode auront bien souvent en commun l'existence des « variables d'état » qui rendent effectivement compte de l'état du système. Pour ces variables, nous ne intéressons qu'à leurs valeurs, et pas à la suite éventuelle d'opérations qui ont permis de les atteindre, utilisant ainsi le principe d'optimalité que l'on peut énoncer de la façon suivante : dans une séquence optimale de décisions, quelle que soit la première décision prise, les décisions suivantes forment une sous-suite qui est optimale, compte tenu des résultats de la première décision. Remarquer que nous avons déjà rencontré un algorithme faisant appel à cette démarche : l'algorithme de Bellman, pour déterminer des chemins optimaux dans les graphes orientés sans circuit.

Exercice 7 – Programmation dynamique

1. Sur le problème de la partition :

   Instance : étant donnés n objets a_i $(1 \leq i \leq n)$ de poids entiers $p(a_1),\ p(a_2), \ldots,\ p(a_n)$,

   Question : Est-il possible de les partager en deux sous-ensembles de même poids total P ?

   (a) Quelle est la condition nécessaire sur la somme des poids des n objets ?

   (b) Nous allons plonger le problème dans une classe de problèmes dépendant de paramètres dépendant de paramètres et liés par une relation de récurrence. On considère deux entiers i et j avec $1 \leq i \leq n$ et $0 \leq j \leq P$, et l'expression booléenne $T(i,j)$: « étant donnés les i premiers éléments de la famille, il existe un sous-ensemble des ces i éléments de poids j ». On remplit alors ligne par ligne un tableau A, qui contient les valeurs de T dont les colonnes sont indicées par j et les lignes par i.

      i. Donner la formule qui lie la ligne i et $i-1$ et $p(a_i)$.

      ii. Illustrer ce principe avec les données suivantes : $n = 6,\ p(a_1) = 5, p(a_2) = 9,\ p(a_3) = 3,\ p(a_4) = 8,\ p(a_5) = 2,\ p(a_6) = 5$.

      iii. Comment avec le tableau rempli obtient-on les éléments de la partition ?

   (c) Donner la complexité de cet algorithme ?

   (d) Faire des jeux d'essais.

2. Le problème du sac à dos :

   Le problème du sac à dos est une généralisation du problème de la partition, on peut s'inspirer de ce qui a été fait plus haut. Soit (P) le problème du sac à dos (généralisé) qui s'écrit sous la forme

$$\begin{cases} \max z = \sum_{j=1}^{n} u_j.x_j \\ \sum_{j=1}^{n} v_j.x_j \leq V \\ x_j \text{ entier pour tout } j \end{cases}$$

   Nous supposerons que tous les coefficients u_j et v_j sont des entiers strictement positifs. On peut considérer (P) comme un cas particulier de la famille des problèmes $(P_{k,v})$ définis pour $1 \leq k \leq n$ et $0 \leq v \leq V$ par :

$$\begin{cases} \max z = \sum_{j=1}^{k} x_j.u_j \\ \sum_{j=1}^{n} v_j.x_j \leq v \\ x_j \text{ entier pour tout } j \text{ compris entre 1 et } k \end{cases}$$

   On a effet $(P) = P_{n,V}$. On a ainsi réalisé le plongement ; il nous reste à exhiber les relations de récurrence permettant de résoudre les problèmes $(P_{k,v})$ de proche en proche. Appelons $z_{k,v}$ le maximum de $(P_{k,v})$. En posant $z_{0,v} = 0$ pour tout v compris entre 0 et V, il vient

$$z_{k,v} = \max_{x_k \in X_k} \{z_{k-1,v-v_k x_k} + u_k x_k\}$$

   en posant

$$X_k = \{x_k \text{ entier tel que } 0 \leq x_k \leq v/v_k\}$$

   (a) Justifier les formules proposées ci-dessus.

   (b) Illustrer le principe sur un exemple de votre choix.

   (c) Quelle est la complexité de cet algorithme ?

3. Le problème du voyageur de commerce :

   Instance Etant donné un graphe complet valué à n sommets (à chaque arête est associée une longueur),

   Question : on cherche un cycle hamiltonien de longueur minimum de ce graphe, un cycle hamiltonien étant un cycle qui passe une fois et une seule fois par chaque sommet et la longueur d'un cycle étant la somme des longueurs des arêtes le constituant.

   (a) Soit une instance du problème du voyageur de commerce, c'est-à-dire la donnée d'une matrice $n \times n$ despoids P d'un graphe $G = (X, E)$ à n sommets, supposés numérotés de 0 à $n-1$. Pour toute partie S de X contenant le sommet 0, et tout sommet i non dans cette partie, on considère le problème suivant : déterminer une plus courte chaîne du sommet 0 au sommet i passant une fois et une seule fois par tout sommet de S et n'utilisant pas de sommet non dans S en dehors de i : appelons $C(S,i)$ la longueur d'une telle chaîne. Si S ne contient que le sommet 0, on voit qu'on a, pour tout $i \neq 0$; $C(S,i) = P(0,i)$ ce qui nous sera utile pour initialiser la récurrence. Sinon, on a

$$C(S,i) = \min_{k \in S - \{0\}} \{C(S - \{k\}, k) + P(k,i)\} \text{ pour } i \notin S$$

   On a donc établi une relation de récurrence sur la taille de S liant les $C(S,i)$. Quant à notre problème lui-même, il suffit pour le résoudre de déterminer la valeur de $\min_{i \in X - \{0\}}\{C(X - \{i\}, i) + P(i,0)\}$. Nous avons bien ici appliqué les principes de la méthode, en plongeant le problème dans une classe plus générale que nous résolvons en utilisant la relation de récurrence.

   (a) Illustrer la méthode en utilisant la figure 4.

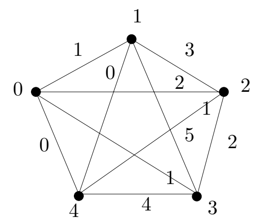

FIG. 4 – Un graphe complet à 5 sommets

(b) Donner la complexité de la méthode. Pour cela évaluer le coût si $|S| = l$. Quel est l'intervalle pour l ?

Exercice 8 – Sur le produit matriciel

Appelons (p,q)-matrice une matrice à p lignes et à q colonnes. Nous admettons que le produit d'une (p,q)-matrice et d'une $(q,r)-$matrice peut se fare à l'aide de pqr opérations.

1. Soit M_i une (p_i, p_{i+1})-matrice, pour $1 \leq i \leq k$. Nous effectuons le produit suivant : $(M_1(M_2(M_3(M_4(\ldots(M_{k-1}.M_k)))))\ldots)$, ainsi de que le produit obtenu en utilisant le parenthésage symétrique $(\ldots(((((M_1.M_2)M_3)M_4)\ldots)M_{k-1})M_k)$ Evaluer le nombre d'opérations nécessaires pour chacun de ces produits. Que remarque-t-'on ?
2. Montrer que le nombre $c(k)$ de parenthésages possibls d'un produit de k matrices vérifie, si on pose $c(1) = 1$, $c(k) = \sum_{i=1}^{k-1} c(i)c(k-i)$. Nous admettrons que $c(k)$ est de l'ordre de $\frac{4^{k-1}}{k\sqrt{\pi k}}$ (ceci peut être obtenu en résolvant la récurrence puis à l'aide de la formule de stirling). Une description exhaustive de tous les parenthésages d'un produit de k matrices, dans le but de déterminer celui qui permet d'effectuer le produit avec un nombre minimum d'opérations, nécessite donc un temps de calcul déraisonnablement exponentiel. La programmation dynamique permet de résoudre ce problème (d'un parenthèsage optimum) en un temps polynomial.
3. En considérant les blocs $M_{ij} = M_i M_{i+j} \ldots M_j$ pour $1 \leq i \leq j \leq k$, et en désignant par m_{ij} le coût minimum du calcul du produit M_{ij}, établir la récurrence permettant d'appliquer le programmation dynamique pour obtenir m_{1k} et le parenthésage correspondant.
4. Appliquer ce principe pour $k = 5, p_1 = 1, p_2 = 2, p_3 = 5, p_4 = 10, p_5 = 4$ et $p_§ = 100$ (nous n'omettrons pas de construire la table correspondante et d'indiquer précisément le parenthésage optimal).

1.5 Méthode Primal-dual

Exercice 9 – Résolution numérique

Résoudre le programme linéaire suivant par la méthode Primal-Dual :

$$Primal \begin{cases} \min z(x_1,x_2,x_3) = 2x_1 + x_2 + 2x_3 + 8x_4 \\ 2x_1 - x_2 + 3x_3 - 2x_4 = 3 \\ -x_1 + 3x_2 - 4x_3 = 1 \\ x_i \geq 0 \end{cases}$$

1.6 Borne de non-approximation

Exercice 10 – Seuil d'approximation pour le problème Bin Packing

Le but de cet exercice est de proposer un seuil d'appproximation pour le problème Bin Packing. Ce résultat sera obtenu à partir du problème Partition.

1. Rappeler le problème Bin packing.
2. Nous partons d'une instance x du problème Partition. Nous construisons une instance x' de Bin packing de la manière suivante :
   – Soit B la somme des éléments de l'instance Partition. Chaque élément a de Partition de poids $p(a) \in \mathbb{N}$, nous contruisons un élément a' de poids $2p(a)/B$.

   Appliquer ce calcul, pour l'instance Partition :

   (a) $x_1 : n = 6,\ p(a_1) = 5, p(a_2) = 9,\ p(a_3) = 3,\ p(a_4) = 8,\ p(a_5) = 2,\ p(a_6) = 5.$

   (b) $x_2 : n = 6,\ p(a_1) = 77, p(a_2) = 41,\ p(a_3) = 3,\ p(a_4) = 30,\ p(a_5) = 17,\ p(a_6) = 12.$
3. Montrer que lorsque nous avons une instance positive alors $m^*(x') = 2$ sinon $m^*(x') = 3$.
4. Quelle est la conséquence sur l'appartenance du problème Bin packing aux classes d'approximation ?

Exercice 11 – Seuil d'approximation pour le problème de la coloration de sommets (resp. d'arêtes)

Dans cet exercice nous nous intéressons à déterminer une borne inférieure pour le problème de la coloratiob de sommets et d'arêtes.

1. Rappeler les résultats de complexité pour ces deux problèmes.
2. Supposons qu'il existe un algorithme A de ayant une performance relative strictement inférieur à 4/3.
   (a) Si $OPT(I) \leq 3$ ($OPT(I)$ désigne la valeur optimale pour une l'instance I quelconque. Conclure pour $A(I)$.

   (b) Si $OPT(I) \geq 4$. Conclure pour $A(I)$.
3. Conclure sur la valeur du seuil d'approximation.

1.7 Comparaison de méthodes

Exercice 12 – Comparaisons branch and bound and branch and cut

Nous considérons le programme linéaire suivant :

$$PL_0 \begin{cases} \max z(x_1,x_2) = 2x_1 + x_2 \\ 2x_1 + 5x_2 \leq 17 \\ 3x_1 + 2x_2 \leq 10 \\ x_1, x_2 \geq 0 \end{cases}$$

1. Représenter graphiquement le polytope associé aux équations de PL_0.
2. Résoudre graphiquement PL_0.
3. Résoudre par la méthode du simplexe. Vous donnerez les tableaux nécessaires à la résolution.
4. Maintenant, nous cherchons une solution à valeur entière :
   (a) Vous donnerez la solution optimale en utilisant la méthode de Branch and Bound. Vous donnerez l'arbre, et vous préciserez toutes les étapes.

   (b) Vous utiliserez la méthode des coupes de Gomory. Vous préciserez à chaque fois la contrainte rajoutée et vous l'exprimerez en fonction (si c'est possible) des variables x_1 et x_2.

2 Partie pratique

Outils :

Pour vous aidez à résoudre vos problèmes de résolution vous pouvez utiliser par exemple l'outil :
- La méthode du simplexe qui se trouve à l'adresse http ://www.zweigmedia.com/RealWorld/simplex.html
- l'outil GLPK (logiciel libre) qui se trouve à l'adresse http ://www.gnu.org/software/glpk/
- ou tout autre solveur

2.1 Programmation dynamique

1. effectuer des simulations pour les problèmes :
   (a) Partition,
   (b) Sac à dos
   (c) Voyageur de commerce

2.2 Branch and Bound

Dans cette partie vous devez programmer la méthode de branch and bound pour résoudre le problème du voyageur de commerce vu en cours. Pour déterminer la solution initiale vous utiliserez :
- la chaîne de poids le plus faible en rajoutant une arête pour obtenir un cycle.
- En utilisant le voisinage $2 - opt$.
- En utilisant le voisinage $3 - opt$.

1. Vous effectuerez des comparaisons sur des jeux d'essais.
2. Programmer l'algorithme ayant une borne supérieure de 3/2 vu en td.
3. Comparer les résultats avec la programmation dynamique.