Algorithmique pour l'optimisation

(Master 1)

Université Claude Bernard Lyon 1

Abstract

1 Optimisation, Recherche Opérationnelle et Modélisation

Pour commencer un cours d'algorithmique sur l'optimisation, il faut déjà se poser la question de qu'est ce que l'optimisation:

> L'optimisation est une branche des mathématiques cherchant à modéliser, à analyser et à résoudre analytiquement ou numériquement les problèmes qui consistent à minimiser ou maximiser une fonction sur un ensemble.
>
> L'optimisation joue un rôle important en recherche opérationnelle, dans les mathématiques appliquées, en analyse et en analyse numérique, en statistique, dans le cadre de la théorie des jeux, ou encore en théorie du contrôle et de la commande.

On voit donc que l'optimisation est une fusée à plusieurs étages:

- D'abord la modélisation dont le but est de transformer un vrai problème de la "vraie vie" en un problème bien posé mathématiquement.
- Ensuite la résolution de ce problème.

On notera que la seconde étape dépend d'une manière cruciale du premier. Autrement dit, lors de l'étape de modélisation, on aura de nombreux choix de modélisation possibles qui ne sont ni bons ni mauvais, juste différents. Le but sera, en fonction des outils mathématiques que l'on souhaite utiliser ensuite (et de notre idée de l'efficacité de ces méthodes pour résoudre nos problèmes), d'utiliser certains de ces objets.

Les problèmes d'optimisation sont très nombreux. Ici on se concentrera sur des problèmes motivés par des problèmes de recherche opérationnelle en général. Selon Wikipedia, la recherche opérationnelle est l'utilisation de modèle mathématiques, statistiques et algorithmiques pour l'aide à la décision. La recherche opérationnelle consiste en l'utilisation de nombreux outils pour rationnaliser, simuler, optimiser et planifier des tâches.

En général, la première étape consiste à modéliser (mathématiquement) le problème qui nous a été donné. Il s'agit ensuite de résoudre le problème mathématique sous-jacent le plus efficacement possible et si possible optimalement (ou le cas échéant essayer d'obtenir une solution aussi proche de l'optimal que possible). Ensuite, on retransforme cette solution en une solution du problème "réel" d'origine et on vérifie que la solution en est bien une. (Que le problème a bien été modélisé, et, surtout, que l'utilisateur final a donné un bon cahier des charges pour ses contraintes.

Une fois que la modélisation est effectuée, on va développer des méthodes algorithmiques pour résoudre les différents problème. On souhaite que les méthodes algorithmiques que l'on développe aient de bonnes propriétés. En particulier:

- Efficace. On doit pouvoir trouver de manière rapide une solution au problème.
- Robuste. Si les données du problème change légèrement (ou que les données obtenus sont bruitées), on doit pouvoir garantir que la solution reste proche d'une solution optimale.

- Garantie théorique ou expérimentale. La solution obtenue doit être la meilleure possible. Si possible optimale, sinon proche de l'être. Si aucune garantie n'existe il faut essayer de construire des heuristiques les plus efficaces possibles en pratique.

Quelques exemples. Quelques exemples de problème de Recherche Opérationnelle:

- **Problèmes de transport.** On se donne n entrepôts $x_1, \ldots, x_n$ tel que chaque x_i a une quantité c_i de ressource et m clients $y_1, \ldots, y_m$ tel que y_i a un besoin d_j en la ressource. Pour chaque paire i, j, il y a un coût de transport de x_i à y_j qui est de $h_{ij} \in \mathbb{R}$ par unité.
  Le but est de satisfaire tous les clients tout en minimisant le coût global de transport.

- **Ordonnancement.** Etant données n tâches $c_1, \ldots, c_n$ qui doivent être faites au cours d'un intervalle $[a_i, b_i]$, on se demande combien on peut en effectuer au maximum sachant que chaque tâche ne peux pas être "décomposée" (soit on la fait entièrement, soit on ne la fait pas du tout, on n'a pas intérêt à n'en faire qu'une partie).

- **Optimisation d'un problème de production.** Avec des contraintes sur les ressources que l'on a et sachant que l'on peut produire plusieurs produits, quel est le mix produit optimal?

- **Classification et clustering.** On se donne un jeu de données X qui sont représentés par des points dans un espace 2-dimensionnel ($\mathbb{R}^2$). Pour évaluer la proximité entre deux points, on utilise la métrique ℓ_2 usuelle du plan. On dit que deux points sont *similaires* si leur distance est au plus un. Le but est de trouver le plus grand *cluster* dans X, c'est à dire le sous ensemble maximum de X de points qui sont deux à deux à distance au plus deux.

Ces quatres exemples se représentent naturellement comme des *programmes linéaires*. Un programme linéaire est un problème d'optimisation d'une fonction linéaire sous des contraintes elles-même linéaires. Les problèmes ci-dessus peuvent être réprésentés comme des programmes linéaires, soit en nombre réel (les variables vont être des variables dans $\mathbb{R}$), soit en nombre entiers. Nous allons voir dans la suite du cours un moyen de résoudre un programme linéaire en nombre réel via *l'algorithme du Simplexe*. Il s'agit d'un algorithme central pour la résolution de problème réel. Cet algorithme, proposé par Dangzig en 1947, est encore utilisé massivement aujourd'hui.

Les problèmes d'optimisation, comme nous le verrons couvrent un champ bien plus vaste que ces problèmes. Par exemple:

- Dons d'organes. Comment affecter optimalement les organes des donneurs aux receveurs?

- Affectation en école (eg Parcours Sup). On cherche des affectations "stables" où, étant donné une affectation, aucun étudiant n'aurait préféré une école différente de celle qu'il a obtenu si l'école le préférrait lui aussi à certains de ces candidats. L'algorithme mis en place en général est souvent l'algorithme de Gale-Shapley (Shapley a obtenu le prix Nobel d'économie, entre autres, pour ces travaux).

Qu'est ce que résoudre un problème d'optimisation. Imaginons qu'on ait un problème à résoudre. Dans un monde parfait, on voudrait concevoir un algorithme extremement rapide qui résout optimalement le problème (et mieux encore qui soit capable de nous donner une garantie que la solution est optimale). Malheureusement la situation n'existe pas forcement. Durant toute la durée du cours, on passera notre temps à devoir faire des concessions entre les facteurs suivants:

- Temps d'execution. On veut un algorithme qui résout rapidement le problème. Dans certains cas, une solution doit être retournée en une fraction de secondes (véhicules autonomes). Parfois plusieurs heures sont acceptables (algorithmes de prédiction du temps)

- Qualité de la solution. Est-elle optimale ? Si non, peut-on donner une garantie (i.e. prouver) sur la qualité de la solution? Parfois une solution optimale est nécessaire (systeme critique dans l'aéronautique) parfois une solution approchée, voir une heuristique sans garantie de performance convient.

- Certificat. Cette question sera moins primordiale dans le cadre du cours. (On concevra les algorithmes dont on prouvera la correction), mais parfois, pour l'utilisateur final pour qui l'algorithme n'est qu'une boite noire, être capable de dire pourquoi ce résultat est optimal peut être important (e.g. "Pourquoi je vois cet publicité").

2 Généralités sur les problèmes d'optimisation

2.1 Problèmes d'optimisation

Dans les prochains cours (comme dans les précédents), on va considérer des problèmes d'optimisation. Un *problème d'optimisation* est un problème de la forme

$$\text{maximize } f(x) \text{ soumis à } x \in F$$

où

- $F \subseteq \mathbb{R}^n$ est appelé *l'ensemble possible*.
- $f : F \to \mathbb{R}$ est appelée la *fonction objectif*.
- x est un vecteur de *varibles de décision*.

Les problèmes d'optimisation apparaissent partout et vous en avez déjà vu des milleurs (que ce soit en optimisation discrète ou continue). Ainsi les problèmes de plus court chemin, de compression, d'optimisation continue, de production industrielle sont tous des problèmes d'optimisation.

Comme vous l'avez déjà vu également, quand on veut résoudre des problèmes d'optimisation, on va parfois regarder quelles sont les propriétés vérifiées par notre problème. En particulier, quand on résout un problème on se rend parfois compte que sa résolution permet en fait de résoudre bien plus que le problème lui même parce qu'on sait résoudre des problèmes bien plus génériques. Cette remarque, à priori inutile sauf pour des vues théoriques à première vue (pourquoi résoudre un problème que je ne me suis pas posé), est en fait importante. En effet:

- Il arrive que les problèmes que l'on ait à résoudre soient légèrement modifiée dans le temps, la méthode que l'on a conçu fonctionne-t-elle toujours?
- Si je suis capable de concevoir une méthode générique pour résoudre des problèmes alors il y a de bonne chance que je puisse résoudre de nombreux problèmes qui pourraient intéresser d'autres industriels.

Notons que si on remplace f par $-f$ on a un problème de minimisation plutot que de maximisation. Donc maximiser $f(x)$ soumis à $x \in F$ correspond à minimiser $-f(x)$ soumis à $x \in F$. Quand F est vide, on dit que le problème n'est *pas réalisable*. Dans de nombreux cas trouver l'existence d'une solution est aussi difficile que de trouver une solution optimale alors que dans d'autres, l'existence d'une solution est triviale. Cela dépend beaucoup du problème qu'on tente de résoudre des contraintes associées.

Quand un vecteur x satisfait toutes les contraintes, on dit qu'il est *réalisable*. Un vecteure x est une *solution optimale* si $f(x)$ est maximisé sur l'ensemble des solutions réalisables. On dit que $f(x)$ est *la valeur optimale*. Quand le problème n'est pas fini (l'ensemble réalisable n'est pas fini) il peut arriver que la valeur optimale soit infinie si, pour tout C, il existe $x \in F$ tel que $f(x) > C$. On note alors $+\infty$ la valeur optimale. Dans la cas d'un problème de minimisation on peut avoir une valeur optimale de $-\infty$.

2.2 Programmes linéaires, quadratiques et au-delà

Supposons qu'on ait un problème d'optimisation de la forme

$$\max f(x)$$

$$F = \{x \in \mathbb{R}^n \text{ tel que } \forall j \leq n, x_j \geq 0 \text{ et } \forall i \leq m, f_i(x) \leq b_i\}$$

Si la fonction f ainsi que toutes les fonctions f_i sont linéaires alors on dit que le problème d'optimisation est un *Programme Lineaire* (ou PL). En d'autres termes, un PL est un problème d'optimisation d'une fonction linéaire sous des contraintes elles-même linéaires. Si au moins une de ces fonctions n'est pas linéaire il s'agit d'un problème d'optimisation non linéaire.

Quand une fonction f_i est linéaire, alors on peut écrire $f_i(x)$ comme $f_i(x) = \sum_{j \leq n} a_{i,j} x_i$ où les $a_{i,j}$ sont des réels et n est la tailel de x (i.e. la taille de la dimension du problème). On peut donc voir chaque f_i comme le produit scale entre un vecteur a_i de taille n et le vecteur de variables x. Notez également que les contraintes de positivité sont des contraintes linéaires (avec le vecteur a qui est nul sauf sur une coordonnée).

On denote par A la matrice où la i-ème ligne est le vecteur a. Autrement dit chaque ligne correspond au vecteur a_i. La matrice A est appelée *matrice de contraintes*. On note b le vecteur $(b_1, \ldots, b_m)$. Le vecteur b est appelé vecteur de contraintes. On peut similairement définir un vecteur c appelé *vecteur objectif* qui correspond aux coefficients de la fonction $f(x)$ que l'on optimise. Notez que m et n ne sont pas nécessairement les mêmes. n est la taille de l'espace sur lesquels on tente d'optimiser notre fonction (le nombre de variables) alors que m correspond au nombre de contraintes. Finalement un programme linéaire peut être résumé de la façon suivante:

Soit A une matrice de taille $m \times n$, $c \in \mathbb{R}^n$ et $b \in \mathbb{R}^n$. Un programme linéaire est un problème d'optimisation de la forme:

$$\begin{aligned} &\text{maximiser}_{x \in \mathbb{R}^n} c^T x \\ &\text{soumis} \\ &Ax \leq b \text{ et} \\ &x_j \geq 0 \text{ pour tout } j \leq m \end{aligned}$$

Dans la suite A sera toujours la matrice de contraintes, b le vecteur de contraintes et c le vecteur objectif.

Si les fonctions ne sont pas linéaires, on peut quand même espérer certaines propriétés: sont-elles quadratiques? Si oui on peut adopter le même genre d'approche en ayant au lieu d'un problème de la forme $Ax \leq b$, on peut espérer avoir un problème bilinéaire de la forme $y^t Ax \leq b$. Si la matrice A est définie positive on parle de programmation semi-définie. Globalement, plus les fonctions deviennent générales, plus on perd de la structure, des informations et la capacité de résoudre nos problèmes !

Optimisation Discrète vs Optimisation Continue. La *Programmation Lineaire en Nombre Entiers* est une classe importante de problème d'optimisation où les fonctions f_i and f sont linéaires mais on optimise selon un vecteur x qui peut seulement prendre des valeurs entières (et pas des valeurs réelles). Autrement dit, the domaine de notre problème devient un sous-ensemble de $\mathbb{Z}^n$ plutot qu'un sous ensemble de $\mathbb{R}^n$. Plus formellement c'est un problème de la forme:

$$\text{maximize } {}_{x \in \mathbb{Z}^n} c^T x \text{ subject to } x \in \mathbb{R}^n \text{ such that } Ax \leq b \text{ and } x_j \geq 0 \text{ for every } j$$

Notez que l'optimum d'un problème de maximisation est toujours plus grand on considère une optimisation en nombre réel plutot qu'en nombre entiers (une solution réelle étant entière). Et l'inverse est vrai pour les problèmes de minimisation. On peut se demander si les valeurs optimales sont toujours les mêmes. La réponse est sans surprise non. On verra par la suite que la situation peut en fait être particulièrement mauvaise quand on passe d'un problème d'optimisation réel à un problème d'optimisation entier.

Si certaines variables sont réelles et d'autres sont entières, on parle de *Programme Linéaire Mixte* (MILP). Dans le cadre de ce cours on abordera simplement les programme linéaires en nombre réels et ceux en nombre entiers mais on ne parlera de techniques particulières pour résoudre les MILP.

Remarque 1. *Soit P un programme linéaire où on tente de minimiser une fonction:*

- *La valeur de la solution optimale réelle au plus égale à la valeur de la solution optimale entière.*
- *L'écart entre la valeur de la solution optimale réelle et entière peut être arbitrairement grand.*

3 Comment modéliser un problème?

Il existe de nombreuses façons de modéliser un problème. Dans le cadre du cours nous allons nous concentrer seulement sur deux d'entre elles:

- **Modélisation via des graphes** (graphes de conflits, graphes de proximité...etc...). Le problème d'origine va se transformer en la recherche de certains objets dans un graphe. Et le but sera de développer des algorithmes efficaces pour trouver ces objets dans les graphes.
- **Modélisatin via des Programmes Linéaires** (en nombre réels ou entiers).

3.1 Modélisation via PL

Mix industriel L'entreprise 'Dovetail' produit deux types d'allumettes, les longues et les courtes. L'entreprise fait un profit de $300\$$ par lot de $100,000$ boites de grandes allumettes et $200\$$ pour $100,000$ boîtes de petites allumettes. L'entreprise a une machine qui lui permet d'empaqueter grandes et petite allummettes qui peut, au total, empaqueter 9 ($\times 100,000$) boites par an. Pour la production d'allumettes, il faut du bois, $3n^3$ pour $100,000$ boites de grandes allumettes et $1m^3$ pour les petites. Au total, l'entreprise a en stock 18 mètres cube de bois. De plus elle a 7 ($\times 100,000$) boites (vides) de grandes alumettes et 6 ($\times 100,000$) de petites. Le but est maximiser le profit l'année suivante.

Quand on veut représenter le problème comme un PL, on doit d'abord trouver les variables. C'est souvent "ce qu'on veut compter". Ici le nombre de petites et grandes boites produites.

- x_1 est le nombre ($\times 100,000$) de grandes boites produites.
- x_2 is the number ($\times 100,000$) de petites boites produites.

Comme le but est de maximiser le produit, on désire:

$$\max_{(x_1,x_2)\in\mathbb{R}^2} 300 \cdot x_1 + 200 \cdot x_2.$$

Maximiser cette fonction est equivalent à maximiser la même fonction où on multiplie les coefficients par le même $\lambda > 0$. Donc on veut maximiser:

$$d = \max_{(x_1,x_2)\in\mathbb{R}^2} 3 \cdot x_1 + 2 \cdot x_2.$$

Maintenant, les contraintes sont: La contrainte de bois

$$3 \cdot x_1 + x_2 \leq 18. \tag{1}$$

La contrainte machine

$$x_1 + x_2 \leq 9. \tag{2}$$

Les contraintes sur les quantités de grandes et petites boites:

$$x_1 \leq 7. \tag{3}$$

$$x_2 \leq 6. \tag{4}$$

Finalement, le nombre de boites est positif donc:

$$x_1, x_2 \geq 0$$

Remarques.

- Le programme est ici vraiment simpliste. Dans la "vraie" vie, il faut imaginer des millers de varialbes et des millions de contraintes. On sait résoudre les PL en nombre réel jusqu'à des ordres de grandeurs qui vont au million de contraintes. Pour les PLNE on s'arrete à la dizaine de milliers seulement ! (Avec beaucoup d'effort !)

- Notez qu'ici on n'a pas dit si les variables étaient entières ou réelles. En fait, on va supposer qu'elles sont réelles. C'est souvent le cas pour des problèmes de mix industriel où la quantité optimale à produire est de l'ordre de milliers voir de millions d'unités. Arrondir la solution optimale à l'entier inférieur laisse en général un probleme satisfiable et une solution "quasi-optimale". Ce n'est pas le cas pour d'autres problèmes, en particulier les problèmes de décision (ie. que l'on peut representer avec des variables dans $\{0,1\}$) puisqu'arrondir toutes les variables à 0 si elles ne valent pas 1 donne en général une solution désastreuse !

Problèmes de transport. On se donne n entrepôts $x_1, \ldots, x_n$ tel que chaque x_i a une quantité c_i de ressource et m clients $y_1, \ldots, y_m$ tel que y_i a un besoin d_j en la ressource. Pour chaque paire i, j, il y a un coût de transport de x_i à y_j qui est de $h_{ij} \in \mathbb{R}$ par unité.

On désire donc modéliser ce problème comme un programme linéaire. Pour cela, il faut d'abord trouver les variables du problème. Les variables sont "ce sur quoi on a du controle", "ce qu'on peut décider de faire ou ne pas faire". Ici la seule marge de manoeuvre est "veut-on livrer une certaine quantité de ressource depuis l'entrepot x_i au client y_j. On va donc créer, pour tout $i \leq n, j \leq m$, une variable $z_{i,j}$ correspond à la quantité de ressource livré de l'entrepot i au client j.

Quels sont les contraintes du problème maintenant? Qu'est ce qu'on doit satisfaire? On doit satisfaire la demande de chaque client. Autrement dit: pour tout $j \leq m$, on veut satisfaire:

$$\sum_{i=1}^{n} z_{i,j} \geq d_j$$

Et comme on ne peut pas livrer des quantités négative de ressource, on satisfait bien sûr: pour tout i, j, $z_{i,j} \geq 0$ (et $z_{ij} \in \mathbb{N}$ si la ressource est discrète).

Pour conclure il suffit d'énoncer la fonction objectif:

$$\min \sum_{i,j} h_{i,j} z_{i,j}.$$

Ordonnancement. Etant données n tâches $C_1, \ldots, C_n$ qui doivent être faites au cours d'un intervalle $[a_i, b_i]$, on se demande combien on peut en effectuer au maximum sachant que chaque tâche ne peux pas être "décomposée".

On désire donc modéliser ce problème comme un programme linéaire. Encore une fois, il faut d'abord trouver les variables du problème, ce sur quoi on a du contrôle. Ici, notre seul choix est le suivant: pour chaque tâche, on peut décider de faire ou de ne pas faire cette tâche. Donc on va créer une variable x_i pour chaque tâche C_i. Chaque variable va prendre la valeur 0 (on ne fait pas la tâche) ou 1 (on effectuera la tâche). Autrement dit, on rajoute la contrainte: $x_i \in \{0,1\}$ pour tout i.

Pour les contraintes, il y a une façon naturelle de réprésenter les contraintes: si les intervalles de C_i et de C_j s'intersectent alors on ne peut pas selectionner les deux tâches. Autrement dit: pour tout i, j tels que C_i et C_j s'intersectent on a $x_i + x_j \leq 1$. Cette contrainte permet de compléter la modélisation du problème.

!!! Attention !!! Ici on peut se rendre compte qu'il existe une autre façon de représenter ce problème... En effet, si on considère la droite réelle et l'ensemble des a_i, b_i, alors on remarque ces ai_i, b_i coupent la droite réelle en au plus $2n+1$ intervalles $I_1, \ldots, I_r$. Et dans chaque intervalle (autrement dit entre deux a_i/b_i consécutifs), l'ensemble des tâches qui contiennent cet intervalle sont les mêmes.

Plutot que de faire la modélisation suivante, on peut donc plutot remarquer que, de tous les intervalles qui contiennent I_ℓ, on peut effectuer au plus l'un d'entre eux (sinon on effectuerai deux tâches au cours de l'intervalles I_ℓ ce qui est impossible). Si on note $\mathcal{C}_\ell$ l'ensemble des tâches qui contiennent I_ℓ, les contraintes peuvent devenir: pour tout $\ell \leq r$,

$$\sum_{i \in \mathcal{C}_\ell} x_i \leq 1$$

A priori, vous direz que ça ne change pas grand chose mais:

- On peut remarquer que le nombre de contraintes ici est linéaire au pire (alors qu'il pouvait être quadratique dans la version précédentes. Quand on sait que la rapidité des algorithmes dépend fortement du nombre de contraintes, l'impact en terme de temps d'execution peut être gigantesque (imaginez si vous avez un million de tâches).
- La deuxième modélisation permet une résolution en temps polynomial alors que la première ne le permet pas ! Cela dépasse le cadre du cours, mais celà illustre l'importance de la phase de modélisation ! Une modélisation ou une autre peut donner des résultats algorithmes très différents !

3.2 Modélisation via des graphes

Ordonnancement. Le problème d'ordonnancement que l'on a discuté ci-dessus peut facilement se représenter sous forme de graphes de conflit. Dans un *graphe de conflit*, deux sommets sont adjacents si ils ne peuvent pas être tous les deux sélectionnés. On considère le graph de conflit suivant. Pour toute tâche C_i, on crée un sommet v_i. Et on rajoute une arête entre v_i et v_j si les intervalles des tâches C_i et C_j s'intersectent.

Le but est maintenant de trouver le plus grand ensemble possible de sommets qui sont deux à deux non adjacents dans le graphe.

!!! Attention !!! On voit encore une fois que la modélisation n'est pas unique. Un problème peut être à la fois modélisé comme un probleme de graphes ou comme un probleme de PL...et certainement comme de nombreux autres problèmes d'optimisation. Le but est de trouver la modélisation la plus appropriée...

Clustering. On se donne un jeu de données X qui sont représentés par des points dans un espace 2-dimensionnel ($\mathbb{R}^2$). Pour évaluer la proximité entre deux points, on utilise la métrique ℓ_2 usuelle du plan. On dit que deux points sont *similaires* si leur distance est au plus un. Le but est de trouver le plus grand *cluster* dans X, c'est à dire le sous ensemble maximum de X de points qui sont deux à deux à distance au plus deux.

L'idée ici est de faire comme dans le cas précédent sauf qu'on regarde le graphe de similarité plutot que le graphe de conflits.

4 Résolution de programmes linéaires en nombre réels

Résolution graphiques On peut résoudre géométriquement le PL en particulier quand la dimension est basse. Voilà la figure ci dessous qui modélise les contraintes.

Remarque 2. *Tout sommet dans un hyperplan normal au vecteur objectif a la même valeur pour la fonction objectif.*

En conséquence, le but est de trouver l'hyperplan le "plus grand" qui intersecte l'ensemble réalisable du programme linéaire.

Algorithmique du Simplexe L'idée est de voyager sur les points extremes du polytope. (voir cours)

Programmes linéaires en nombre entiers. Résoudre de tels programmes linéaires est beaucoup plus compliqué ! En fait, il est NP-complet en général de trouver une solution optimale d'un programme linéaire en nombres entiers (et même approximer une solution est difficile).

On verra cependant plus tard dans le cours des techniques algorithmiques qui ne sont malheureusement pas aussi génériques pour résoudre de tels programmes linéaires.

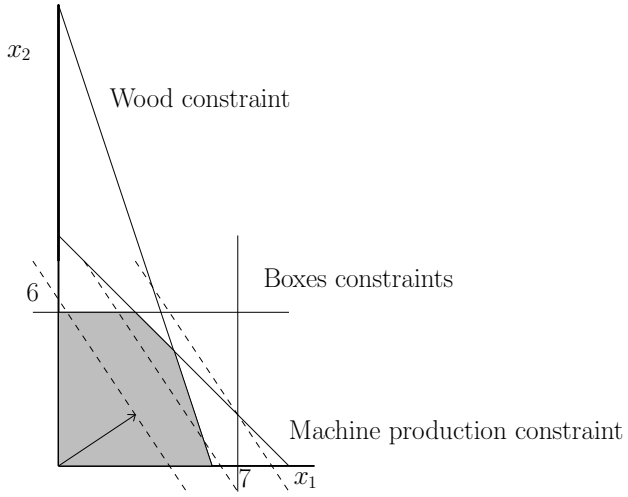

Figure 1: Geometrical representation of the Dovetail LP. In dotted, lines with the same value for the objective function. The gray part is the feasible region.