

Gestion de projet : Construction d'un stade

Florian Dargel

November 2021

1 Introduction

On souhaite résoudre un problème de programmation linéaire en utilisant IBM CPLEX OPTIMIZER qui est un solveur de programmation mathématique hautes performances pour programmation linéaire, programmation mixte en nombre entier et programmation par contraintes. On définira le sujet qui concerne la construction d'un stade puis la méthode pour résoudre notre problème.

2 Construction d'un stade

2.1 Le sujet

À partir des données du fichier Stade.ods, on souhaite construire un stade dans les meilleurs délais. On doit proposer des dates de réalisation des différentes tâches de façon à minimiser la durée totale du projet. On expliquera le modèle et commentera la solution obtenue.

Dans notre base de données (Image 1 ci-dessous), on a :

- Le numéro des tâches i : 19 numéros
- Le nom des tâches : installation de chantier, terrassements...
- La durée d'une tâche p_i : on peut exprimer cette durée en semaines.
- Les tâches précédentes : certaines tâches ne peuvent être exécutées avant la fin d'autres tâches.

N° tâche (i)	Libellé des tâches	Durée p(i)	Tâches précédentes
1	Installation du chantier	2	Aucune
2	Terrassements	16	1
3	Construction des fondations	9	2
4	VRD (voirie, réseaux divers)	8	2
5	Élévation du sous-sol	10	3
6	Plancher principal	6	4, 5
7	Cloisonnement des vestiaires	2	4
8	Electrification des gradins	2	6
9	Pose du toit	9	4, 6
10	Eclairage du stade	5	4
11	Installation des gradins	3	6
12	Mise hors d'eau du toit	2	9
13	Finition des vestiaires	1	7
14	Construction de la billetterie annexe	7	2
15	Voirie secondaire	4	4, 14
16	Signalétique	3	8, 11, 14
17	Pelouse et accessoires sportifs	9	12
18	Réception de l'ouvrage	1	17
19	Fin des travaux	0	Toutes

Image 1 : Base de données de construction d'un stade

2.2 La formulation du programme linéaire

On a un **problème d'ordonnancement**. Les tâches sont indicées par i allant de 1 à 19. On définit p_i la durée de la tâche i . On peut représenter notre problème sous forme de graphe (Image 2 ci-dessous) pour gérer les précédences entre les tâches. Soit $G=(X,U)$, défini par l'ensemble des tâches X et un ensemble d'arcs U : un arc (i,j) signifie que la tâche i doit précéder la tâche j .

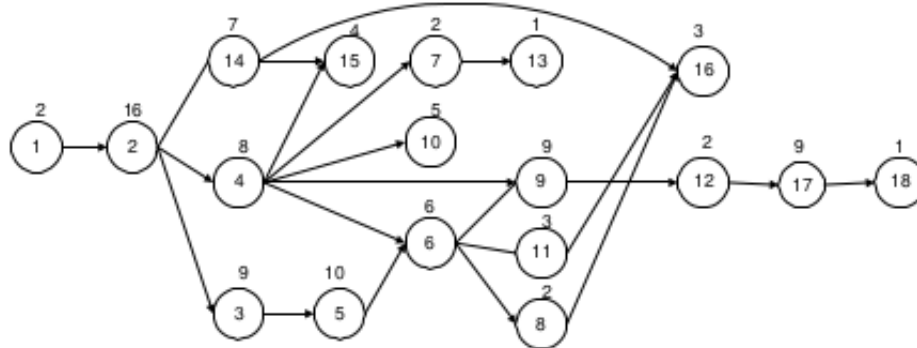


Image 2 : Graphe des précédences

On définit la variable t_i pour les dates de début des tâches, comptées à partir du temps zéro. Les seules contraintes à respecter sont les contraintes de précédence. Une tâche j ne peut démarer que si toutes les tâches qui doivent la précéder sont terminées, s'il existe un arc de i à j , alors la date de fin de i ($t_i + p_i$) ne doit pas dépasser la date de début de j .

Notre objectif à ce problème est de minimiser t_{19} qui correspond à la date de la fin des travaux. On définit donc le modèle suivant :

$$\begin{cases} \text{Min } t_{19} \\ \forall (i,j) \in U : t_i + p_i \leq t_j \\ \forall i = 1 \dots 19 : t_i \geq 0 \end{cases}$$

2.3 Résolution du problème linéaire avec CPLEX

Le programme linéaire que l'on veut résoudre avec CPLEX doit être dans un fichier texte. Le nom d'un fichier texte contenant notre programme linéaire au format LP doit finir par ".lp". J'ai nommé mon fichier "test.lp". Voici la description de notre programme au format LP :

```

minimize
Nom_objectif : t19
st
Nom_contrainte0 : t1 = 0
Nom_contrainte1 : t1 - t2 <= -2
Nom_contrainte2 : t2 - t3 <= -16
Nom_contrainte3 : t2 - t4 <= -16
Nom_contrainte4 : t3 - t5 <= -9
Nom_contrainte5 : t4 - t6 <= -8
Nom_contrainte6 : t5 - t6 <= -10
Nom_contrainte7 : t4 - t7 <= -8
Nom_contrainte8 : t6 - t8 <= -6
Nom_contrainte9 : t4 - t9 <= -8
Nom_contrainte10 : t6 - t8 <= -6
Nom_contrainte11 : t4 - t9 <= -8
Nom_contrainte12 : t6 - t9 <= -6
Nom_contrainte13 : t4 - t10 <= -8
Nom_contrainte14 : t6 - t11 <= -6
Nom_contrainte15 : t9 - t12 <= -9
Nom_contrainte16 : t7 - t13 <= -2
Nom_contrainte17 : t2 - t14 <= -16
Nom_contrainte18 : t4 - t15 <= -8

```

```

Nom_contrainte19 : t14 - t15 <= -7
Nom_contrainte20 : t8 - t16 <= -2
Nom_contrainte21 : t11 - t16 <= -3
Nom_contrainte22 : t14 - t16 <= -7
Nom_contrainte23 : t12 - t17 <= -2
Nom_contrainte24 : t17 - t18 <= -9
Nom_contrainte25 : t1 - t19 <= -2
Nom_contrainte26 : t2 - t19 <= -16
Nom_contrainte27 : t3 - t19 <= -9
Nom_contrainte28 : t4 - t19 <= -8
Nom_contrainte29 : t5 - t19 <= -10
Nom_contrainte30 : t6 - t19 <= -6
Nom_contrainte31 : t7 - t19 <= -2
Nom_contrainte32 : t8 - t19 <= -2
Nom_contrainte33 : t9 - t19 <= -9
Nom_contrainte34 : t10 - t19 <= 5
Nom_contrainte35 : t11 - t19 <= -3
Nom_contrainte36 : t12 - t19 <= -2
Nom_contrainte37 : t13 - t19 <= -1
Nom_contrainte38 : t14 - t19 <= -7
Nom_contrainte39 : t15 - t19 <= -4
Nom_contrainte40 : t16 - t19 <= -3
Nom_contrainte41 : t17 - t19 <= -9
Nom_contrainte42 : t18 - t19 <= -1
bounds
t1 >=0
t2 >=0
t3 >=0
t4 >=0
t5 >=0
t6 >=0
t7 >=0
t8 >=0
t9 >=0
t10 >=0
t11 >=0
t12 >=0
t13 >=0
t14 >=0
t15 >=0
t16 >=0
t17 >=0
t18 >=0
t19 >=0
end

```

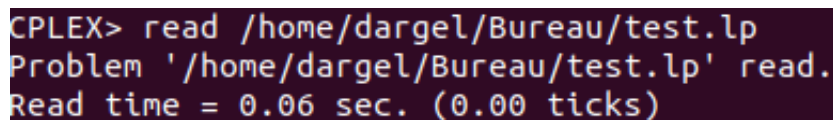
Minimize : Définit la fonction objectif (ou fonction économique) à minimiser.

st : Définit les contraintes de notre modèle.

bounds : Contient uniquement les contraintes de bornes.

end : La fin de notre programme

On commence par lire notre fichier "test.pl" sur CPLEX avec la commande "**read**".



```

CPLEX> read /home/dargel/Bureau/test.lp
Problem '/home/dargel/Bureau/test.lp' read.
Read time = 0.06 sec. (0.00 ticks)

```

Il nous donne comme information que notre fichier s'est lu en 0.06 secondes

On peut voir les informations de notre problème avec la commande "**disp pr st**"

```

CPLEX> disp pr st
Problem name       : /home/dargel/Bureau/test.lp
Objective sense    : Minimize
Variables          :      19
Objective nonzeros :      1
Linear constraints  :      43 [Less: 42, Equal: 1]
  Nonzeros         :      85
  RHS nonzeros     :      42

Variables          : Min LB: 0.000000      Max UB: all infinite
Objective nonzeros : Min   : 1.000000      Max   : 1.000000
Linear constraints  :
  Nonzeros         : Min   : 1.000000      Max   : 1.000000
  RHS nonzeros     : Min   : 1.000000      Max   : 16.000000

```

CPLEX nous présente un problème avec 43 contraintes linéaires et 19 variables.

On résoud ensuite notre problème d'optimisation avec la commande **"optimize"**.

```

CPLEX> optimize
Version identifier: 20.1.0.0 | 2020-11-10 | 9bedb6d68
Tried aggregator 1 time.
LP Presolve eliminated 38 rows and 14 columns.
Aggregator did 5 substitutions.
All rows and columns eliminated.
Presolve time = 0.00 sec. (0.02 ticks)

Dual simplex - Optimal: Objective = 6.400000000000e+01
Solution time = 0.02 sec. Iterations = 0 (0)
Deterministic time = 0.04 ticks (2.38 ticks/sec)

```

CPLEX prétraite les problèmes en simplifiant les contraintes, en réduisant la taille des problèmes et en éliminant la redondance. Son présolveur (LP Presolve) essaie de réduire la taille d'un problème en faisant des déductions sur la nature de toute solution optimale au problème. Son agrégateur essaie d'éliminer les variables et les lignes par substitution (aggregator did 5 substitutions).

On affiche ensuite les valeurs de toutes les variables dans la solution optimale de notre programmation linéaire qui vient d'être résolu, en utilisant la commande **display solution variable** -

```

CPLEX> display solution variables -
Variable Name          Solution Value
t19                    64.000000
t2                     2.000000
t3                    18.000000
t4                    18.000000
t5                    27.000000
t6                    37.000000
t7                    26.000000
t8                    43.000000
t9                    43.000000
t10                   59.000000
t11                   43.000000
t12                   52.000000
t13                   63.000000
t14                   18.000000
t15                   60.000000
t16                   61.000000
t17                   54.000000
t18                   63.000000
All other variables in the range 1-19 are 0.

```

Les solutions de la variable t_i pour les dates des débuts des tâches.

On observe que la construction du stade peut se terminer en 64 semaines minimum, on a $t_{19} = 64$.

On a des tâches qui peuvent se réaliser au même moment (à la même semaine), on peut réaliser une frise chronologique et observer à quel moment s'effectue une tâche (image 3 : durée de la construction du stade).

Soit $n = \{2, 18, 26, 27, 37, 43, 52, 54, 59, 60, 61, 63, 64\}$

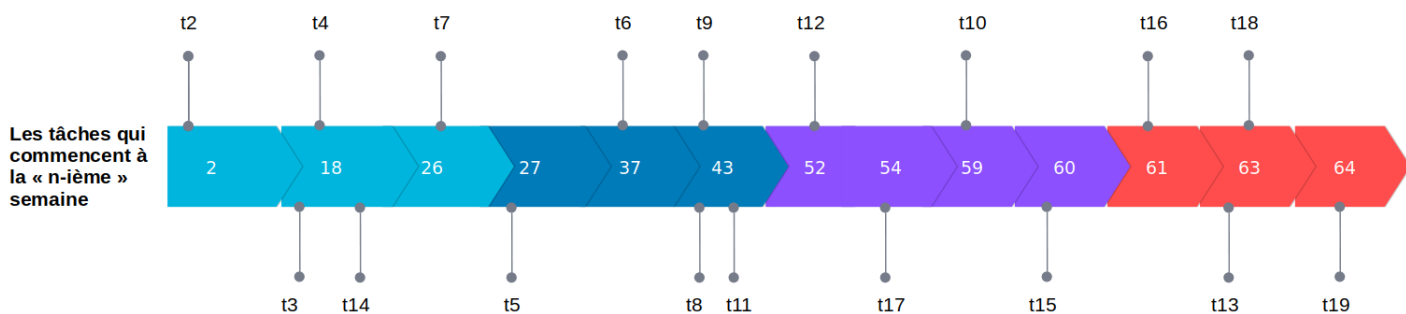


Image 3 : durée de la construction du stade

3 Conclusion :

On a résolu un problème de programmation linéaire en utilisant un solveur de programmation mathématique hautes performances pour programmation linéaire, programmation mixte en nombre entiers et programmation par contraintes. A partir des données du fichier Stade.ods, on a proposé des dates de réalisation des différentes tâches de façon à minimiser la durée totale du projet. La durée totale du projet pour construire notre stade est de 64 semaines.