Ecole Supérieure
d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

COMPTE RENDU

Mathematics for Engineering - TP2
3e année Cybersécurité - École Supérieure d'Informatique et du Numérique (ESIN)
Collège d'Ingénierie & d'Architecture (CIA)

Étudiant : HATHOUTI Mohammed Taha

Filière : Cybersecurité

Année : 2025/2026

Enseignants : Mme.OUFASKA

Date : 23 novembre 2025

Table des matières

1 Exercice 1 : Affectation d'ouvriers à des tâches

Une entreprise de construction doit affecter 4 ouvriers à 4 tâches. Le tableau ci-dessous indique l'efficacité de la personne si elle est affectée à la tâche. Une barre (–) indique que la personne n'est pas qualifiée pour la tâche.

	Tâche 1	**Tâche** 2	**Tâche** 3	**Tâche** 4
Ouvrier 1	45	–	–	30
Ouvrier 2	50	55	15	–
Ouvrier 3	–	60	25	75
Ouvrier 4	45	–	–	35

TABLE 1 – Matrice d'efficacité des ouvriers

Objectif : Maximiser l'efficacité totale en affectant chaque ouvrier à exactement une tâche et chaque tâche à exactement un ouvrier, en respectant les qualifications.

Modélisation mathématique

Ensembles :

— $O = \{O_1, O_2, O_3, O_4\}$: ensemble des ouvriers

— $T = \{T_1, T_2, T_3, T_4\}$: ensemble des tâches

Paramètres :

— Eff_{ij} : efficacité de l'ouvrier i pour la tâche j

— $Qual_{ij} \in \{0, 1\}$: vaut 1 si l'ouvrier i est qualifié pour la tâche j, 0 sinon

Variables de décision :

— $x_{ij} \in \{0, 1\}$: vaut 1 si l'ouvrier i est affecté à la tâche j, 0 sinon

Fonction objectif :

$$\text{Maximiser} \quad Z = \sum_{i \in O} \sum_{j \in T} Eff_{ij} \cdot x_{ij} \tag{1}$$

Contraintes :

$$\sum_{j \in T} x_{ij} = 1, \quad \forall i \in O \quad \text{(chaque ouvrier affecté à une seule tâche)} \tag{2}$$

$$\sum_{i \in O} x_{ij} = 1, \quad \forall j \in T \quad \text{(chaque tâche reçoit un seul ouvrier)} \tag{3}$$

$$x_{ij} \leq Qual_{ij}, \quad \forall i \in O, \forall j \in T \quad \text{(respect des qualifications)} \tag{4}$$

$$x_{ij} \in \{0, 1\}, \quad \forall i \in O, \forall j \in T \tag{5}$$

1.1 Ecrire ce programme mathématique en OPL dans fichier *affectation.mod*

```
/***** affectation.mod *****/

/* Ensembles */
{string} Ouvriers = ...;
{string} Taches = ...;
int Eff[Ouvriers][Taches] = ...;
int Qual[Ouvriers][Taches] = ...; // Valeurs 0/1

/* Variables de decision : x[i][j] = 1 si l'ouvrier i est affecte
    a la tache j */
dvar boolean x[Ouvriers][Taches];

/* Objectif : maximiser l'efficacite totale */
maximize
  sum (i in Ouvriers, j in Taches) Eff[i][j] * x[i][j];

/* Contraintes */
subject to {
  /* (1) Chaque ouvrier est affecte a exactement une tache */
  forall (i in Ouvriers)
    sum (j in Taches) x[i][j] == 1;

  /* (2) Chaque tache recoit exactement un ouvrier */
  forall (j in Taches)
    sum (i in Ouvriers) x[i][j] == 1;

  /* (3) Interdictions (barres "--") : on force x[i][j] = 0 si
     non qualifie */
  forall (i in Ouvriers, j in Taches)
    x[i][j] <= Qual[i][j];
}
```

Listing 1 – Modèle OPL pour l'affectation

1.2 Ecrire un fichier *affectation.dat* représentant les données.

```
/***** affectation.dat *****/

Ouvriers = {"O1", "O2", "O3", "O4"};
Taches = {"T1", "T2", "T3", "T4"};

/* Matrice d'efficacite (Eff[i][j]) */
Eff = [
  [45,  0,  0, 30],
  [50, 55, 15,  0],
  [ 0, 60, 25, 75],
  [45,  0,  0, 35]
```

```
];

/* Matrice de qualification (Qual[i][j])
   1 = l'ouvrier est qualifie, 0 = non qualifie */
Qual = [
  [1, 0, 0, 1],
  [1, 1, 1, 0],
  [0, 1, 1, 1],
  [1, 0, 0, 1]
];
```

Listing 2 – Données pour l'affectation

1.3 Résoudre cette instance sous OPL Studio.

```
// solution (optimal) with objective 160
// Quality Incumbent solution:
// MILP objective                                1,6000000000e+02
// MILP solution norm |x| (Total, Max)           4,00000e+00  1,00000e
   +00
// MILP solution error (Ax=b) (Total, Max)       0,00000e+00  0,00000e
   +00
// MILP x bound error (Total, Max)               0,00000e+00  0,00000e
   +00
// MILP x integrality error (Total, Max)         0,00000e+00  0,00000e
   +00
// MILP slack bound error (Total, Max)           0,00000e+00  0,00000e
   +00

x = [[1 0 0 0]
     [0 1 0 0]
     [0 0 1 0]
     [0 0 0 1]];
```

2 Exercice 2 : Problème diététique

Il s'agit d'un régime alimentaire garantissant un apport suffisant en éléments nutritifs. On considère :

- n aliments au prix unitaire de c_j $(j = 1, \ldots, n)$
- m éléments nutritifs
- q_{ij} la quantité du $i^{\text{ème}}$ élément nutritif contenue dans une unité du $j^{\text{ème}}$ aliment
- d_i quantité minimale requise de l'élément nutritif i $(i = 1, \ldots, m)$
- x_j variable de décision représentant la quantité de l'aliment j à acheter

Une modélisation mathématique du problème est :

$$\begin{aligned} \text{Minimiser} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{s.c. :} \quad & \sum_{j=1}^{n} q_{ij} x_j \geq d_i, \quad \forall i \in \{1, \ldots, m\} \\ & x_j \geq 0, \quad j \in \{1, \ldots, n\} \end{aligned}$$

2.1 Ecrire ce programme mathématique en OPL dans fichier *diet.mod*

```
/***** diet.mod *****/

/* Parametres */
int m = ...; /* nombre d'elements nutritifs */
int n = ...; /* nombre d'aliments */
range Nutritifs = 1..m;
range Aliments = 1..n;
float c[Aliments] = ...; /* prix unitaire des aliments */
float q[Nutritifs][Aliments] = ...; /* quantite de nutriment i
   dans aliment j */
float d[Nutritifs] = ...; /* quantite minimale requise de
   nutriment i */

/* Variables de decision */
dvar float x[Aliments]; /* quantite d'aliment j a acheter */

/* Fonction objectif : minimiser le cout total */
minimize
  sum(j in Aliments) c[j] * x[j];

/* Contraintes */
subject to {
  /* Contrainte de satisfaction des besoins nutritionnels */
  forall(i in Nutritifs)
    sum(j in Aliments) q[i][j] * x[j] >= d[i];

  /* Contrainte de positivite */
  forall(j in Aliments)
    x[j] >= 0;
}
```

Listing 3 – Modèle OPL pour le problème diététique

2.2 Ecrire un fichier *diet.dat* représentant les données

On prend $m = 3$, $n = 3$, $c = [15, 17, 16]$, $q = [[8, 13, 7], [12, 7, 3], [12, 21, 8]]$, $d = [8, 7, 6]$.

```
/***** diet.dat *****/

/* Nombre d'elements nutritifs */
m = 3;

/* Nombre d'aliments */
n = 3;

/* Prix unitaire de chaque aliment (c[j]) */
c = [15, 17, 16];

/* Matrice des quantites nutritionnelles (q[i][j])
   q[i][j] = quantite du nutriment i contenue dans une unite de l
      'aliment j */
q = [
  [8, 13, 7],   /* Nutriment 1 dans chaque aliment */
  [12, 7, 3],   /* Nutriment 2 dans chaque aliment */
  [12, 21, 8]   /* Nutriment 3 dans chaque aliment */
];

/* Quantites minimales requises de chaque nutriment (d[i]) */
d = [8, 7, 6];
```

Listing 4 – Données pour le problème diététique

2.3 Résoudre cette instance sous OPL Studio

```
// solution (optimal) with objective 12.05
// Quality There are no bound infeasibilities.
// There are no reduced-cost infeasibilities.
// Max. unscaled (scaled) Ax-b resid.          = 8,88178e-16 (5,55112e
   -17)
// Max. unscaled (scaled) c-B'pi resid.        = 8,88178e-16 (8,88178e
   -16)
// Max. unscaled (scaled) |x|                  = 0,4 (0,4)
// Max. unscaled (scaled) |slack|              = 6,6 (0,4125)
// Max. unscaled (scaled) |pi|                 = 7,3 (15,84)
// Max. unscaled (scaled) |red-cost|           = 0 (0)
// Condition number of scaled basis            = 8,2e+00

x = [0.35
     0.4
     0];
```

3 Exercice 3 : Problème de transport

Une entreprise dispose de quatre entrepôts ($E1$, $E2$, $E3$, $E4$) pour des unités destinées à satisfaire la demande de quatre clients ($C1$, $C2$, $C3$, $C4$). Le nombre d'unités disponibles à chaque entrepôt et les demandes des clients sont spécifiés dans le tableau suivant qui contient également le coût du transport d'un item de chaque entrepôt à chaque client.

	Client 1	**Client 2**	**Client 3**	**Client 4**	**Disponibilité**
Entrepôt 1	5	6	4	2	10
Entrepôt 2	2	300	1	3	20
Entrepôt 3	3	4	2	1	20
Entrepôt 4	2	1	3	2	10
Demande	20	10	10	20	

TABLE 2 – Coûts de transport, disponibilités et demandes

3.1 Ecrire ce programme mathématique en OPL dans fichier *transport1.mod*

```
/***** transport1.mod *****/

/* Parametres */
int nbEntrepots = ...;
int nbClients = ...;
range Entrepots = 1..nbEntrepots;
range Clients = 1..nbClients;
float cout[Entrepots][Clients] = ...; /* cout de transport de l'
   entrepot i vers le client j */
float stock[Entrepots] = ...; /* disponibilite a chaque entrepot
   */
float demande[Clients] = ...; /* demande de chaque client */

/* Variables de decision */
dvar float x[Entrepots][Clients]; /* quantite transportee de l'
   entrepot i vers le client j */

/* Fonction objectif : minimiser le cout total de transport */
minimize
  sum(i in Entrepots, j in Clients) cout[i][j] * x[i][j];

/* Contraintes */
subject to {
  /* Contrainte de capacite : ne pas depasser le stock disponible
      a chaque entrepot */
  forall(i in Entrepots)
    sum(j in Clients) x[i][j] <= stock[i];

  /* Contrainte de demande : satisfaire la demande de chaque
     client */
  forall(j in Clients)
```

```
27      sum(i in Entrepots) x[i][j] >= demande[j];
28
29    /* Contrainte de positivite */
30    forall(i in Entrepots, j in Clients)
31      x[i][j] >= 0;
32  }
```

Listing 5 – Modèle OPL pour le problème de transport

3.2 Ecrire un fichier *transport1.dat* représentant les données

```
1  /***** transport1.dat *****/
2
3  /* Nombre d'entrepots */
4  nbEntrepots = 4;
5
6  /* Nombre de clients */
7  nbClients = 4;
8
9  /* Matrice des couts de transport (cout[i][j])
10    cout[i][j] = cout de transport d'une unite de l'entrepot i
          vers le client j */
11 cout = [
12   [5, 6, 4, 2],       /* Entrepot 1 vers chaque client */
13   [2, 300, 1, 3],     /* Entrepot 2 vers chaque client */
14   [3, 4, 2, 1],       /* Entrepot 3 vers chaque client */
15   [2, 1, 3, 2]        /* Entrepot 4 vers chaque client */
16 ];
17
18 /* Disponibilite a chaque entrepot (stock[i]) */
19 stock = [10, 20, 20, 10];
20
21 /* Demande de chaque client (demande[j]) */
22 demande = [20, 10, 10, 20];
```

Listing 6 – Données pour le problème de transport

3.3 Résoudre cette instance sous OPL Studio

```
// solution (optimal) with objective 100
// Quality There are no bound infeasibilities.
// There are no reduced-cost infeasibilities.
// Maximum Ax-b residual              = 0
// Maximum c-B'pi residual            = 0
// Maximum |x|                       = 10
// Maximum |slack|                   = 10
// Maximum |pi|                      = 299
// Maximum |red-cost|                = 0
// Condition number of unscaled basis = 4,9e+01

x = [[0  0  0 10]
```

```
    [10  0  10   0]
    [10  0   0  10]
    [0  10   0   0]];
```

4 Exercice 4 : Problème du voyageur de commerce - Formulation MTZ

La formulation MTZ (Miller-Tucker-Zemlin) du problème du voyageur de commerce (TSP) peut être présentée comme suit :

On considère les variables de décision :

— x_{ij} : variable binaire égale à 1 si l'arc (i, j) fait partie du tour du voyageur, 0 sinon

— u_i : variable entière qui définit l'ordre dans lequel la ville i est visitée

$$
\begin{aligned}
\text{Minimiser} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \\
\text{s.c. :} \quad & \sum_{j=1}^{n} x_{ij} = 1, \quad \forall i \in \{1,\dots,n\} \\
& \sum_{i=1}^{n} x_{ij} = 1, \quad \forall j \in \{1,\dots,n\} \\
& u_i - u_j + (n-1)x_{ij} \leq n-2, \quad \forall i,j \in \{2,\dots,n\} \\
& 1 \leq u_i \leq n, \quad \forall i \in \{1,\dots,n\} \\
& x_{ij} \in \{0,1\}
\end{aligned}
$$

4.1 Ecrire ce programme mathématique en OPL dans fichier *tspmtz.mod*

```
/***** tspmtz.mod *****/

/* Parametres */
int n = ...; /* nombre de villes */
range Villes = 1..n;
range Villes2 = 2..n; /* villes sans la ville de depart (ville 1) */
float d[Villes][Villes] = ...; /* matrice des distances d[i][j] */

/* Variables de decision */
dvar boolean x[Villes][Villes]; /* x[i][j] = 1 si l'arc (i,j) fait partie du tour, 0 sinon */
dvar int u[Villes]; /* u[i] = ordre de visite de la ville i */

/* Fonction objectif : minimiser la distance totale du tour */
minimize
  sum(i in Villes, j in Villes) d[i][j] * x[i][j];
```

```
16
17 /* Contraintes */
18 subject to {
19   /* (1) Chaque ville doit avoir exactement une sortie */
20   forall(i in Villes)
21     sum(j in Villes: j != i) x[i][j] == 1;
22
23   /* (2) Chaque ville doit avoir exactement une entree */
24   forall(j in Villes)
25     sum(i in Villes: i != j) x[i][j] == 1;
26
27   /* (3) Contrainte MTZ pour eviter les sous-tours */
28   forall(i in Villes2, j in Villes2: i != j)
29     u[i] - u[j] + (n - 1) * x[i][j] <= n - 2;
30
31   /* (4) Bornes sur les variables u */
32   forall(i in Villes)
33     u[i] >= 1;
34
35   forall(i in Villes)
36     u[i] <= n;
37 }
```

Listing 7 – Modèle OPL pour TSP - Formulation MTZ

4.2 Calculer en fonction de n le nombre de contraintes et le nombre de variables de décisions de cette formulation

Nombre de variables de décision :

- Variables x_{ij} (binaires) : $n \times n = n^2$ variables
- Variables u_i (entières) : n variables
- **Total :** $n^2 + n$ **variables**

Nombre de contraintes :

- Contraintes de sortie : n contraintes
- Contraintes d'entrée : n contraintes
- Contraintes MTZ : $(n-1)(n-2)$ contraintes (pour $i, j \in \{2, \ldots, n\}, i \neq j$)
- Bornes inférieures sur u_i : n contraintes
- Bornes supérieures sur u_i : n contraintes
- **Total :** $n + n + (n-1)(n-2) + n + n = n^2 + n + 2$ **contraintes**

Application numérique pour bays29 ($n = 29$) **:**

- Variables : $29^2 + 29 = 841 + 29 = 870$ variables
- Contraintes : $29^2 + 29 + 2 = 841 + 29 + 2 = 872$ contraintes

4.3 Résoudre l'instance bays29

Fichier de données : *tspmtz.dat* contient la matrice de distances 29×29 pour l'instance *bays*29 (29 villes en Bavière, Allemagne).

```
// solution (optimal) with objective 2027
// Quality Incumbent solution:
// MILP objective                                  2,0270000000e+03
// MILP solution norm |x| (Total, Max)             4,36000e+02  2,80000e
   +01
// MILP solution error (Ax=b) (Total, Max)         0,00000e+00  0,00000e
   +00
// MILP x bound error (Total, Max)                 0,00000e+00  0,00000e
   +00
// MILP x integrality error (Total, Max)           0,00000e+00  0,00000e
   +00
// MILP slack bound error (Total, Max)             0,00000e+00  0,00000e
   +00

x = [[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0]
     [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
     [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
     [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
     [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
     [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]];

u = [1 19 21 14 23 25 4 27 24 15 6 26 17 8 13 7 11 10 12 16 18 9 3 1 5
   22 2 28 20];
```

5 Exercice 5 : Problème du voyageur de commerce - Formulation SSB

La formulation SSB (Sarin-Sherali-Bhootra) du problème du voyageur de commerce (TSP) peut être présentée comme suit :

On considère les variables de décision :

— x_{ij} : variable binaire égale à 1 si l'arc (i, j) fait partie du tour du voyageur, 0 sinon

— u_{ij} : variable binaire égale à 1 si la ville i précède (pas nécessairement immédiatement) la ville j, 0 sinon

$$
\begin{aligned}
\text{Minimiser} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij} \\
\text{s.c. :} \quad & \sum_{\substack{j=1 \\ j\neq i}}^{n} x_{ij} = 1, \quad \forall i \in \{1,\dots,n\} \\
& \sum_{\substack{i=1 \\ i\neq j}}^{n} x_{ij} = 1, \quad \forall j \in \{1,\dots,n\} \\
& u_{ij} \geq x_{ij}, \quad \forall i,j \in \{2,\dots,n\}, i \neq j \\
& u_{ij} + u_{ji} = 1, \quad \forall i,j \in \{2,\dots,n\}, i \neq j \\
& u_{ij} + u_{jk} + u_{ki} \leq 2, \quad \forall i,j,k \in \{2,\dots,n\}, i \neq j, j \neq k \\
& x_{ij}, u_{ij} \in \{0,1\}
\end{aligned}
$$

5.1 Ecrire ce programme mathématique en OPL dans fichier *tspssb.mod*

```
/***** tspssb.mod *****/

/* Parametres */
int n = ...;
range Villes = 1..n;
range Villes2 = 2..n;
float d[Villes][Villes] = ...;

/* Variables de decision */
dvar boolean x[Villes][Villes];
dvar boolean u[Villes2][Villes2];

/* Fonction objectif */
minimize
  sum(i in Villes, j in Villes: i != j) d[i][j] * x[i][j];

/* Contraintes */
subject to {
  /* (1) Chaque ville doit avoir exactement une sortie */
```

```
  forall(i in Villes)
    sum(j in Villes: j != i) x[i][j] == 1;

  /* (2) Chaque ville doit avoir exactement une entree */
  forall(j in Villes)
    sum(i in Villes: i != j) x[i][j] == 1;

  /* (3) Si on va directement de i vers j, alors i precede j */
  forall(i in Villes2, j in Villes2: i != j)
    u[i][j] >= x[i][j];

  /* (4) Entre deux villes i et j, l'une precede forcement l'
     autre */
  forall(i in Villes2, j in Villes2: i < j)
    u[i][j] + u[j][i] == 1;

  /* (5) Contrainte de transitivite */
  forall(i in Villes2, j in Villes2, k in Villes2: i != j && j !=
      k && i != k)
    u[i][j] + u[j][k] + u[k][i] <= 2;
}
```

Listing 8 – Modèle OPL pour TSP - Formulation SSB

5.2 Calculer en fonction de n le nombre de contraintes et le nombre de variables de décisions et faire une comparaison avec la formulation MTZ

Nombre de variables de décision SSB :

- Variables x_{ij} (binaires, $i \neq j$) : $n(n-1)$ variables
- Variables u_{ij} (binaires, $i, j \in \{2, \ldots, n\}$) : $(n-1)^2$ variables
- **Total :** $n(n-1) + (n-1)^2 = n^2 - n + n^2 - 2n + 1 = 2n^2 - 3n + 1$ **variables**

Nombre de contraintes SSB :

- Contraintes de sortie : n contraintes
- Contraintes d'entrée : n contraintes
- Contraintes de précédence : $(n-1)(n-2)$ contraintes
- Contraintes d'ordre : $\frac{(n-1)(n-2)}{2}$ contraintes
- Contraintes de transitivité : $(n-1)(n-2)(n-3)$ contraintes
- **Total :** $2n + (n-1)(n-2) + \frac{(n-1)(n-2)}{2} + (n-1)(n-2)(n-3)$ **contraintes**

Application numérique pour bays29 ($n = 29$) :

Formulation SSB :

- Variables : $29 \times 28 + 28^2 = 812 + 784 = 1596$ variables
- Contraintes : $2 \times 29 + 28 \times 27 + \frac{28 \times 27}{2} + 28 \times 27 \times 26 = 58 + 756 + 378 + 19656 = 20848$ contraintes

Comparaison MTZ vs SSB pour $n = 29$:

Formulation	Variables	Contraintes
MTZ	870	872
SSB	1596	20848

TABLE 3 – Comparaison MTZ vs SSB pour bays29

Observations :

— La formulation SSB utilise environ 1,8 fois plus de variables que MTZ

— La formulation SSB utilise environ 24 fois plus de contraintes que MTZ

5.3 Résoudre l'instance bays29 et comparer les temps de résolution

Fichier de données : *tspssb_bays29.dat* (même matrice de distances que pour MTZ)

```
// solution (optimal) with objective 2027
// Quality Incumbent solution:
// MILP objective                                  2,0270000000e+03
// MILP solution norm |x| (Total, Max)             4,07000e+02  1,00000e
   +00
// MILP solution error (Ax=b) (Total, Max)         0,00000e+00  0,00000e
   +00
// MILP x bound error (Total, Max)                 0,00000e+00  0,00000e
   +00
// MILP x integrality error (Total, Max)           0,00000e+00  0,00000e
   +00
// MILP slack bound error (Total, Max)             0,00000e+00  0,00000e
   +00

x = [[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
     [0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0]
     [0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
     [0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0]
     [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
     [0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]]
```

```
    [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0]
    [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
    [0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]];
```

Comparaison des résultats :

Formulation	**Objectif obtenu**	**Temps (sec)**
MTZ	2027	0,52
SSB	2027	3,68
Optimal connu	**2020**	–

TABLE 4 – Comparaison des performances pour bays29

Observations :

— Les deux formulations (MTZ et SSB) ont trouvé la même solution avec un objectif de 2027

— Cette solution est proche de l'optimal connu (2020)

— La formulation MTZ est environ 7 fois plus rapide que SSB (0,52 sec vs 3,68 sec)

— Malgré ses contraintes plus fortes, SSB n'a pas permis d'atteindre l'optimal dans ce cas

— Pour l'instance bays29, MTZ s'avère plus efficace en termes de temps de calcul tout en donnant la même qualité de solution