



# MODÉLISATION MATHÉMATIQUE ET OPTIMISATION DES FLUX D'ÉVACUATION D'UN STADE DE FOOTBALL

## Rapport final MAP05

26 avril 2024

---

Tanguy Azéma, Sophie Carlier, Matthieu Hautsch, Arthur Iffenecker, Lucas Mebille, Félix Rosseeuw

# TABLE DES MATIÈRES

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Modélisation mathématique</b>	<b>4</b>
2.1	Programmation linéaire . . . . .	4
2.1.1	Définitions . . . . .	4
2.1.2	Propriétés . . . . .	5
2.1.3	Résolution : algorithme du simplexe . . . . .	6
2.1.4	Programmation linéaire en nombres entiers . . . . .	7
2.2	Les problèmes de flots . . . . .	9
2.2.1	Flots dans un graphe . . . . .	9
2.2.2	Problème de flot à coût minimum . . . . .	10
2.3	Notre modèle de stade . . . . .	13
2.3.1	Graphe statique . . . . .	13
2.3.2	Graphe dynamique . . . . .	14
2.3.3	Choix du coût et résolution . . . . .	19
<b>3</b>	<b>Notre démarche avec le stade Auguste Delaune</b>	<b>23</b>
3.1	Observations et prises de mesures . . . . .	23
3.2	Modélisation numérique . . . . .	26
<b>4</b>	<b>Analyse des résultats et simulation de différents scénarios</b>	<b>28</b>
4.1	Cas général et étude tribune par tribune . . . . .	28
4.2	Influence de l'affluence . . . . .	29
4.3	Cas où une sortie est bloquée . . . . .	31
4.4	Cas où un escalier est bloqué . . . . .	32
<b>5</b>	<b>Conclusion</b>	<b>35</b>
<b>A</b>	<b>Annexe</b>	<b>37</b>

# 1 INTRODUCTION

Ces dernières années, la gestion des foules au sein des enceintes sportives, en particulier dans les stades de football, a émergé comme un enjeu de première importance, engendrant de nombreuses problématiques et tensions. L'affluence considérable, variant de 20 000 à 80 000 spectateurs selon les capacités des infrastructures, exige une planification minutieuse des dispositifs de sécurité, de contrôle et de gestion des flux, constituant ainsi un défi majeur pour les responsables de la sécurité des clubs hôtes. L'objectif prépondérant demeure la fluidité de l'événement tout en réduisant les temps d'attente et les congestions aux points d'accès, afin de prévenir tout risque d'incidents graves. L'ampleur des conséquences d'une gestion défaillante des foules s'est tragiquement illustrée lors de la finale de la Ligue des champions 2022 au Stade de France, où des défaillances dans la supervision des supporters ont engendré des scènes chaotiques. Le souvenir d'Hillsborough en 1989, marqué par une bousculade fatale dans une tribune surchargée, demeure un poignant rappel des potentielles conséquences d'une gestion inefficace des foules.

Face à ce défi crucial, nous avons opté pour une approche scientifique afin d'analyser la gestion des foules dans les stades de football. Alors que les stratégies traditionnelles se fondent souvent sur l'expérience et le pragmatisme, notre intention était de rechercher des stratégies scientifiquement fondées et théoriquement optimales. Parmi les différentes approches envisageables, nous avons privilégié l'utilisation des mathématiques appliquées à travers la théorie des flots, particulièrement adaptée à la complexité géométrique des stades et à l'échelle des foules à l'intérieur de ces structures. Nous nous sommes également inspirés de travaux antérieurs sur l'évacuation des grands bâtiments utilisant la théorie des flots.

Dans le cadre de notre étude, nous avons choisi de nous concentrer initialement sur l'évacuation du stade, considérant cette problématique comme particulièrement propice à une modélisation via la théorie des flots et cruciale dans la gestion des foules. Pour ce faire, le Stade Auguste Delaune à Reims a été sélectionné comme terrain d'étude en raison de sa configuration relativement simple et de la disponibilité des données fournies par le club.

Notre projet scientifique visait ainsi à explorer des solutions d'optimisation pour l'évacuation du Stade Auguste Delaune en utilisant la théorie des flots et la bibliothèque NetworkX de Python. Ce rapport détaille les différentes phases de notre démarche, de la modélisation mathématique à l'implémentation numérique, en passant par la collecte de données, avant de présenter nos conclusions sur les méthodes permettant d'améliorer l'évacuation du stade.

## 2 MODÉLISATION MATHÉMATIQUE

### 2.1 PROGRAMMATION LINÉAIRE

Nous posons ici le cadre théorique dans lequel s'inscrit notre modélisation mathématique du problème d'évacuation d'un stade de football. Nous introduisons dans un premier temps la programmation linéaire (PL) qui décrit de façon générique la famille des problèmes d'optimisation auxquels notre problématique appartient.

Dans un second temps, nous nous concentrerons sur les problèmes de flots, que l'on peut résoudre grâce à la programmation linéaire, et qui constituent le cadre formel de notre modélisation.

#### 2.1.1 • DÉFINITIONS

La programmation linéaire (PL) constitue une branche fondamentale de la recherche opérationnelle, visant à résoudre des problèmes d'optimisation où les fonctions objectifs et les contraintes sont linéaires. On peut formaliser un *programme linéaire* de la façon suivante : soit  $A \in \mathbb{R}^{m \times n}$ ,  $A' \in \mathbb{R}^{m' \times n}$  deux matrices,  $b \in \mathbb{R}^m$ ,  $b' \in \mathbb{R}^{m'}$ ,  $c \in \mathbb{R}^n$  des vecteurs colonnes. Le problème d'optimisation étudié est le suivant :

$$\inf_{x \in \mathbb{R}^n, Ax \leq b, A'x = b'} c_1 x_1 + \dots + c_n x_n \quad (1)$$

La matrice  $A$  et le vecteur  $b$  (respectivement la matrice  $A'$  et le vecteur  $b'$ ) représentent les contraintes d'inégalité du problème (respectivement les contraintes d'égalité). Elles prendront un sens physique lorsque nous transposerons plus tard la formalisation des programmes linéaires à notre problème d'évacuation.

Il est facile de montrer que, quitte à augmenter la taille des données  $A \in \mathbb{R}^{m \times n}$  et  $b \in \mathbb{R}^m$  avec  $m \leq n$  et à introduire des variables d'écart positives  $\lambda$  telles que  $Ax = b + \lambda$ , tout problème linéaire du type (1) peut se mettre sous la forme *standard* :

$$\inf_{x \in \mathbb{R}^n, Ax = b, x \geq 0} c_1 x_1 + \dots + c_n x_n \quad (2)$$

De plus on peut supposer que  $\text{rg}(A) = m$  car sinon certaines lignes de  $A$  sont liées, i.e. soit certaines contraintes sont incompatibles et donc il n'y a pas de solutions à (2) soit elles sont redondantes et on peut supprimer des lignes.

**Définition.** L'ensemble  $X_{ad}$  des vecteurs  $x \in \mathbb{R}^n$  satisfaisant les contraintes est appelé ensemble des **solutions admissibles**. De plus, on appelle sommet de  $X_{ad}$  tout point  $x \in X_{ad}$  qui ne s'écrit pas comme une combinaison convexe non triviale de deux points de  $X_{ad}$ .

$X_{ad}$  est un polyèdre (c'est-à-dire une intersection finie de demi-espaces de  $\mathbb{R}^n$ ) convexe.

Notre objectif sera de trouver non pas une solution admissible mais bien une **solution optimale** qui minimise notre fonction objectif que nous préciserons plus loin.

### 2.1.2 • PROPRIÉTÉS

Nous regroupons ici les principales propriétés des solutions d'un programme linéaire standard utiles pour notre projet.

Nous admettons le lemme suivant qui caractérise l'existence de solutions optimales au programme linéaire (2) :

**Lemme 2.1.** *Il existe une solution optimale au programme linéaire (2) dès lors que le minimum de la fonction objectif sur l'ensemble des solutions admissibles  $X_{ad}$  est fini. La réciproque est vraie.*

Afin de justifier plus loin la pertinence de l'algorithme du simplexe que nous utilisons pour résoudre notre problème nous introduisons la décomposition suivante :

**Définition.** On appelle **base** associée à (2) une base de  $\mathbb{R}^m$  formée de  $m$  colonnes de  $A$ . Cette base  $\mathcal{B}$  est une sous-matrice inversible carrée de  $A$ . On peut alors, à une permutation près des colonnes, écrire  $A = (\mathcal{B}, \mathcal{C})$  avec  $\mathcal{C} \in \mathbb{R}^{m \times (n-m)}$ . Ainsi pour tout  $x \in \mathbb{R}^n$  on peut introduire la décomposition  $x = (x_{\mathcal{B}}, x_{\mathcal{C}})$  avec  $x_{\mathcal{B}} \in \mathbb{R}^m$  et  $x_{\mathcal{C}} \in \mathbb{R}^{n-m}$ . On a alors  $Ax = \mathcal{B}x_{\mathcal{B}} + \mathcal{C}x_{\mathcal{C}}$ . Une solution  $x \in X_{ad}$  telle que  $x_{\mathcal{C}} = 0$  est appelée une **solution basique**. Si de plus l'une des composantes de  $x_{\mathcal{B}}$  est nulle, alors on dit que c'est une **solution basique dégénérée**.

**Lemme 2.2.** *Les sommets de  $X_{ad}$  sont les solutions basiques de (2).*

*Démonstration.* Soit  $x$  une solution basique de (2) pour la base  $\mathcal{B}$  formée par ses vecteurs colonnes  $\{b_1, \dots, b_m\}$  de  $\mathbb{R}^m$ . On peut compléter cette famille de vecteurs en une base de  $\mathbb{R}^n$ . Dans cette base complétée on peut écrire  $x = (x_1, \dots, x_m, 0, \dots, 0)^T$  de sorte que  $x_1b_1 + \dots + x_mb_m = b$  car  $x \in X_{ad}$  ( $\mathcal{B}x_{\mathcal{B}} = b$ ). Soit  $y, z \in X_{ad}$  tels que  $x = \theta y + (1 - \theta)z$  avec  $\theta \in [0, 1]$ . Nécessairement les  $(n - m)$  dernières composantes de  $y$  et  $z$  sont nulles et  $y_1b_1 + \dots + y_mb_m = z_1b_1 + \dots + z_mb_m = b$ . Par unicité de la décomposition de  $b$  dans une base, on a  $x = y = z$  et donc  $x$  est un sommet de l'ensemble des solutions admissibles.

Réciproquement, si  $x$  est un sommet de  $X_{ad}$ , soit  $k$  le nombre de ses composantes non nulles, et à une permutation près des composantes on peut écrire  $x = (x_1, \dots, x_k, 0, \dots, 0)^T$  et  $b = x_1a_1 + \dots + x_ka_k$ , où les  $a_i$  sont des colonnes de  $A$ . Montrons alors que ces colonnes forment une famille libre de  $\mathbb{R}^m$  que l'on pourra compléter en une base, si bien que  $x$  sera une solution basique associée à cette dernière. Pour cela supposons que ces colonnes forment une famille liée de  $\mathbb{R}^m$ . Alors il existe  $y \in \mathbb{R}^n$  non nul tel que  $y_1a_1 + \dots + y_ka_k = 0$  et ses  $n - k$  dernières composantes soient nulles. Ainsi par la stricte positivité des  $k$  premières composantes de  $x$  on peut trouver une  $\epsilon > 0$  suffisamment petit tel que  $x + \epsilon y$  et  $x - \epsilon y$  soient des solutions admissibles (car  $A(x + \epsilon y) = A(x - \epsilon y) = Ax = b$  et  $x \pm \epsilon y \geq 0$  pour  $\epsilon$  assez petit). Dès lors  $x$  est le milieu de ces deux points de  $X_{ad}$  ce qui est absurde puisque c'est un sommet. Ainsi les  $k$  colonnes de  $A$  forment une famille libre de  $\mathbb{R}^m$  que l'on peut compléter en une base de (2).  $x$  est donc **une solution basique associée à la base ainsi construite**.  $\square$

On admet le théorème suivant :

**Théorème 2.1.** *Si il existe une solution optimale à un programme linéaire standard alors il existe une solution optimale basique.*

Ainsi, le lemme et le théorème précédents permettent d'établir que nous pouvons nous limiter aux sommets du polyèdre des solutions admissibles pour trouver une solution optimale à notre programme linéaire. Cette propriété fondamentale est exploitée par l'algorithme de résolution que nous présentons plus loin : l'algorithme du simplexe.

Nous terminons sur la démonstration de la condition d'optimalité suivante qui justifiera l'efficacité de l'algorithme du simplexe [1].

**Proposition 2.1.** *Soit une solution basique  $x^* = (x_B^*, 0)$  associée à une base  $\mathcal{B}$ . On décompose le vecteur de coût  $c = (c_B, c_C)$ . Si le vecteur des coûts réduits  $\tilde{c}_C = c_C - \mathcal{C}^T(\mathcal{B}^{-1})^T c_B \geq 0$ , alors  $x^*$  est optimale. Si de plus  $x^*$  est non dégénérée, alors la réciproque est vraie.*

*Démonstration.* Soit  $x \in X_{ad}$ ,  $x = (x_B, x_C)$  avec  $x_B = \mathcal{B}^{-1}(b - \mathcal{C}x_C)$ . Montrons que si le vecteur des coûts réduits est positif alors la valeur de la fonction objectif  $c \cdot x$  en  $x$  est supérieure à sa valeur en  $x^*$ .

On a  $x_B^* = \mathcal{B}^{-1}b$ . Donc  $c \cdot x - c \cdot x^* = c_B \cdot x_B + c_C \cdot x_C - c_B \cdot x_B^* = \tilde{c}_C \cdot x_C$ . Si  $\tilde{c}_C \geq 0$  puisque  $x_C \geq 0$  alors  $c \cdot x - c \cdot x^* \geq 0$  et  $x^*$  est optimal.

Réciroquement si  $\tilde{c}_C$  a une composante strictement négative selon une direction  $e_i$ , on définit  $x(\epsilon) = (x_B(\epsilon), x_C(\epsilon))$  avec :

$$\begin{aligned} x_C(\epsilon) &= \epsilon e_i \\ x_B(\epsilon) &= \mathcal{B}^{-1}(b - \mathcal{C}x_C(\epsilon)) \end{aligned}$$

Si on suppose que  $x^*$  est une solution non-dégénérée, i.e.  $\mathcal{B}^{-1}b > 0$ , alors  $x(\epsilon) \geq 0$  et  $x(\epsilon) \in X_{ad}$  pour  $\epsilon$  assez petit. On désigne par  $\epsilon^*$  un tel  $\epsilon$ .

Enfin, il suffit de voir que  $x(0) = x^*$  et  $c \cdot x(\epsilon^*) < c \cdot x^*$  et donc  $x^*$  n'est pas optimal dans ce cas.  $\square$

### 2.1.3 • RÉSOLUTION : ALGORITHME DU SIMPLEXE

L'algorithme du simplexe, introduit par le mathématicien George Dantzig à partir de 1947, utilise les propriétés précédentes pour trouver une solution optimale à un programme linéaire. Il parcourt les sommets du polyèdre  $X_{ad}$ , qui sont donc les solutions basiques du problème, jusqu'à trouver une solution optimale. Ce parcours est effectué de manière astucieuse en énumérant les sommets tout en faisant décroître la fonction objectif d'un sommet à l'autre.

L'astuce permettant de réduire le coût entre chaque sommet exploré exploite la condition d'optimalité de la proposition 2.1. Voici une description de l'algorithme du simplexe en principe. Si plusieurs variantes existent [2], nous présentons ici la version la plus simple :

- **Initialisation.** On part d'un sommet (donc d'une solution basique)  $x^0$  associée à une base  $\mathcal{B}^0$ .

- **$k$ -ème itération.** On dispose d'un sommet  $x^k$  associée à une base  $\mathcal{B}^k$  et du coût réduit  $\tilde{c}_C^k$  :
  - Si le coût réduit  $\tilde{c}_C^k$  est positif, alors d'après la proposition 2.1  $x^k$  est optimal et c'est fini ;
  - Si on peut vérifier que le problème est non-borné on s'arrête ;
  - Sinon  $\tilde{c}_C^k$  possède une composante strictement négative et nous pouvons trouver un nouveau sommet de coût inférieur. On met alors à jour la base associée  $\mathcal{B}^{k+1}$ .

Nous ne détaillons pas ici comment trouver un sommet de coût inférieur ni comment mettre à jour la base car en pratique nous utiliserons l'algorithme du simplexe en « boîte noire » via NetworkX. Nous nous contenterons des remarques complémentaires suivantes qui nous suffisent pour analyser l'algorithme appliqué à notre problème.

### Remarques.

- Si tous les sommets explorés par l'algorithme du simplexe sont des solutions basiques non-dégénérées, alors l'algorithme converge en un nombre fini d'étapes. Cela est dû au fait que la non-dégénérescence implique une réduction stricte du coût à chaque itération lors du choix d'un nouveau sommet et de la base associée.
- En pratique on observe jamais de *phénomène de cyclage* qui consiste en l'enumération d'une solution dégénérée au cours des itérations. Celle-ci pourrait conduire à la stationnarité de la fonction de coût entre deux itérations et donc nuire à la convergence [1].
- On peut se donner une représentation géométrique très simple de l'algorithme du simplexe (fig. 1) : à chaque itération on passe d'un sommet du polyèdre  $X_{ad}$  à un sommet adjacent en suivant une arête permettant de faire diminuer la fonction de coût. Si on ne rencontre pas de sommet le long de cette arête le problème est non-borné ( $\inf_{x \in X_{ad}} c \cdot x = -\infty$ ).
- Un sommet du polyèdre  $X_{ad}$  est l'intersection de  $n$  égalités parmi les  $n+m$  égalités admises par les contraintes du problème (2) ( $n$  contraintes de positivité et les  $m$  autres contraintes d'égalité), il y a donc un nombre de sommets exponentiel en la dimension du problème. Or Klee et Minty [4] ont montré qu'il existait un programme linéaire pour lequel l'algorithme du simplexe peut dans le pire cas explorer tous les sommets. L'algorithme a donc une **compléxité théorique exponentielle**. Cependant on constate qu'en pratique la résolution d'un programme linéaire ne prend pas plus de quelques minutes et que les pires cas sont très rares. La compléxité observée est en général **polynomiale** [1] [3].

#### 2.1.4 • PROGRAMMATION LINÉAIRE EN NOMBRES ENTIERS

Pour pouvoir appliquer la théorie de la programmation linéaire à notre problème nous avons besoin d'introduire une nouvelle approche à laquelle notre problème pourra facilement s'identifier. Depuis le début nous avons adopté un point de vue continu de notre espace des solutions admissibles en ne le restreignant que via les contraintes de positivité et d'égalité. Or notre problème d'évacuation qui vise à optimiser le flux de personnes dans le stade nécessite de forcer les variables du problèmes - qui représenteront comme nous le verrons des quantités de spectateurs

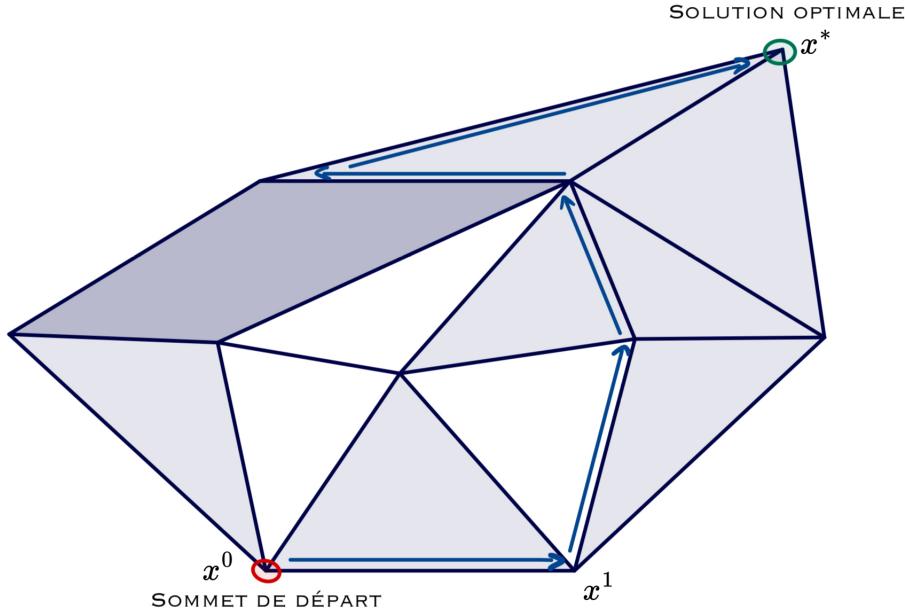


FIGURE 1 – Réprésentation géométrique de la convergence de l'algorithme du simplexe.

- à être entières. Or l'étude des programmes linéaires à variables entières constitue un champ d'étude complet de l'optimisation linéaire appelé *optimisation combinatoire* ou *programmation linéaire en nombres entiers (PLNE)*.

Dans ce cadre là, le programme linéaire standard (2) se reformule de la sorte :

$$\inf_{x \in \mathbb{Z}^n, Ax = b, x \geq 0} c_1 x_1 + \dots + c_n x_n \quad (3)$$

Dans le cas général il est difficile de résoudre un PLNE et le minimum de (3) est à priori plus grand que celui du *problème relaxé* (2). Cependant notre problème entre dans un cas particulier pour lequel il est relativement simple de trouver une solution car les deux minimums coïncident et donc l'algorithme du simplexe fournit une solution à valeurs entières. Cela est dû à une structure particulière des données que nous étudions ici.

**Définition.** Une matrice carrée  $A \in \mathbb{Z}^{n \times n}$  est dite **unimodulaire** dès lors que  $\det A = \pm 1$ . Une matrice rectangulaire  $A \in \mathbb{Z}^{m \times n}$  est dite **totalement unimodulaire** lorsque toute sous-matrice carrée de  $A$  est de déterminant  $\pm 1$  ou 0.

On dispose du lemme de Poincaré qui donne une condition suffisante pour qu'une matrice soit totalement unimodulaire :

**Lemme 2.3.** Soit une matrice  $A$  à coefficients dans  $\{-1, 0, 1\}$ . Si  $A$  a au plus un coefficient 1 et un coefficient  $-1$  par colonne, alors elle est totalement unimodulaire.

*Démonstration.* Les hypothèses sur les coefficients de  $A$  sont vraies pour toute sous matrice de  $A$ . On peut donc se limiter à une matrice carrée  $A$  et vérifier que  $\det A \in \{-1, 0, 1\}$ . Si  $A$  a

une colonne nulle alors son déterminant est nul. Si il existe une colonne qui ne possède qu'un unique coefficient non-nul alors un développement du déterminant par cette colonne donne que  $\det A \in \{-1, 0, 1\}$ . Enfin il reste le cas où chaque colonne possède exactement un 1 et un  $-1$ . Dans ce cas la somme des lignes est nulle et donc le vecteur  $(1, \dots, 1)^T$  appartient au noyau de  $A^T$  qui a donc un déterminant nul, et donc  $A$  aussi. Dans tous les cas  $A$  est donc totalement unimodulaire.  $\square$

Ce type de données est important pour la proposition suivante :

**Proposition 2.2.** *On suppose que les données  $A$  et  $b$  de (3) sont entières et que  $A$  est totalement unimodulaire. Alors toute solution basique  $x$  est entière.*

*Démonstration.* En s'appuyant sur la décomposition introduite plus haut, une solution basique associée à une base  $\mathcal{B}$  est un  $x = (x_{\mathcal{B}}, x_{\mathcal{C}})$  avec  $x_{\mathcal{C}} = 0$ ,  $x_{\mathcal{B}} \geq 0$  et  $\mathcal{B}x_{\mathcal{B}} = b$ . Or  $A$  est totalement unimodulaire donc  $\mathcal{B}$  est unimodulaire et on peut montrer facilement que  $\mathcal{B}^{-1}b \in \mathbb{Z}^m$ . Donc  $x = (x_{\mathcal{B}}, 0) \in \mathbb{Z}^n$ .  $\square$

La proposition (2.2) assure que si notre problème respecte les hypothèses énoncées - nous verrons plus loin pourquoi elles le sont dans notre cas - alors l'algorithme du simplexe qui énumère des solutions basiques et qui trouve une solution basique optimale retournera une solution optimale entière de (3). Attention toutes les solutions ne sont à priori pas entières. Puisque la fonction objectif et les contraintes sont linéaires il est possible que plusieurs solutions basiques optimales existent et donc par linéarité toute combinaison convexe de celles-ci est solution.

Nous avons donc posé le cadre théorique des méthodes d'optimisation que nous allons utiliser pour notre projet. Il nous faut maintenant trouver une structure de données permettant de représenter rigoureusement un stade de football et quantifier la circulation des spectateurs en son sein. Cette structure doit aussi être compatible avec l'approche par programmation linéaire que nous venons d'étudier. Pour cela nous avons choisi d'utiliser la notion de flots dans un graphe que nous présentons maintenant.

## 2.2 LES PROBLÈMES DE FLOTS

---

### 2.2.1 • FLOTS DANS UN GRAPHE

Nous rappelons tout d'abord la définition d'un graphe.

**Définition.** Soit  $V$  un ensemble et  $E$  une partie de  $V \times V$ . L'ensemble  $G = (V, E)$  est alors un **graphe**.  $V$  désigne l'ensemble des sommets, aussi appelés **noeuds** et  $E$  l'ensemble des **arcs** ou des arêtes. Dans le cadre de notre projet les graphes considérés seront finis, c'est-à-dire que  $E$  sera un ensemble fini et  $V$  aussi. On note  $m$  le nombre de noeuds et  $n$  le nombre d'arêtes. De plus les graphes considérés sont orientés c'est à dire que si  $(a, b) \in E$ , nous n'avons pas forcément  $(b, a) \in E$ . On appelle **chemin** dans un graphe une suite d'arcs successifs reliant deux nœuds. Un chemin est **élémentaire** si tous ces noeuds sont distincts. Un **circuit** est une suite successive d'arêtes distinctes dont les nœuds de départ et d'arrivée sont identiques.

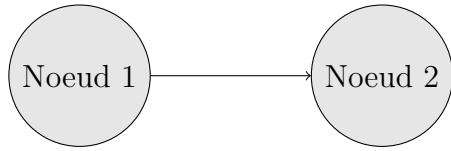


FIGURE 2 – Illustration d'un graphe orienté à deux noeuds

Nous pouvons maintenant introduire la notion de flots dans un graphe qui est fondamentale pour notre problème.

**Définition.** Soit  $G = (V, E)$  un graphe orienté. Chaque arc  $(i, j) \in E$  possède une capacité maximale  $u_{ij} \in \mathbb{R}_+ \cup \{+\infty\}$  et un coût  $c_{ij} \in \mathbb{R}$ . De plus, chaque nœud du graphe  $i \in V$  est associé à un **flot entrant exogène**  $b_i \in \mathbb{R}$ . On appelle alors **flot** un vecteur  $x \in \mathbb{R}^n$  défini sur  $E$ ,  $(i, j) \mapsto x_{ij}$  vérifiant les lois suivantes :

La **loi des nœuds de Kirchhoff** :

$$b_i + \sum_{\substack{j \in V \\ (j, i) \in E}} x_{ji} = \sum_{\substack{j \in V \\ (i, j) \in E}} x_{ij}, \quad \forall i \in V \quad (4)$$

La **contrainte de positivité** :

$$0 \leq x_{ij}, \quad \forall (i, j) \in E \quad (5)$$

On dit alors qu'un flot est **admissible** lorsqu'il vérifie en plus de (4) et (5) :

$$x_{ij} \leq u_{ij}, \quad \forall (i, j) \in E \quad (6)$$

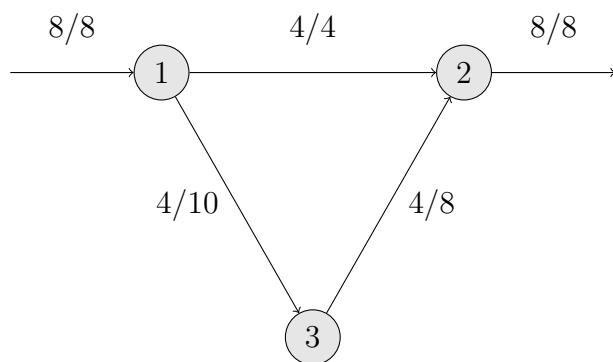


FIGURE 3 – Exemple de graphe à 3 noeuds, avec le couple flot/capacité  $(x_{i,j}/u_{i,j})$  sur les arêtes.

### 2.2.2 • PROBLÈME DE FLOT À COÛT MINIMUM

En considérant cette structure de données, on appelle problème de flot à coût minimum le programme linéaire :

$$\min_{x \in \mathbb{R}^n} \sum_{(i, j) \in E} c_{ij} x_{ij} \quad (7)$$

Où le flot  $x \in \mathbb{R}^n$ ,  $(i, j) \mapsto x_{ij}$  est un flot admissible.

Enfin la proposition suivante est cruciale pour justifier l'interprétabilité physique d'un flot dans un graphe dans le cadre de notre problématique d'évacuation.

**Définition.** Un **flot unitaire associé à un chemin** est un flot  $x$  tel que  $x_{ij} = 1$  pour tout  $(i, j)$  appartenant à ce chemin, et  $x_{ij} = 0$  pour  $(i, j)$  n'appartient pas à ce chemin.

**Proposition 2.3.** Dans un graphe  $G$  défini comme ci-dessus, ne contenant pas de cycle, tout flot admissible  $x$  se décompose en une combinaison linéaire positive de flots unitaires associés à des chemins élémentaires d'un noeud entrée vers un noeud sortie.

Ainsi pour un flot représentant une évacuation des spectateurs, chaque flot unitaire de la décomposition précédente représente le chemin d'un individu d'une entrée vers une sortie du stade.

*Démonstration.* Soit  $x$  un flot admissible, non identiquement nul. On appelle un **noeud entrée**, un noeud  $i \in V$  où  $b_i > 0$  et un **noeud sortie** un noeud où  $b_i < 0$ . On appelle le support de  $x$  l'ensemble des arcs  $(i, j) \in E$  tel que  $x_{ij} > 0$ .

Prenons  $(i, j)$  un arc dans le support de  $x$ , on va construire un **chemin contenant l'arc  $(i, j)$  et inclus dans le support de  $x$ , qui relie un noeud entrée à un noeud sortie.**, on a d'après la loi des noeuds en  $j$  :

$$b_j + \sum_{(l,j) \in E} x_{lj} = \sum_{(j,k) \in E} x_{jk}$$

D'après la contrainte (5), et le fait que l'arc  $(i, j)$  soit dans le support de  $x$ , on a :

$$\sum_{(l,j) \in E} x_{lj} \geq x_{ij} > 0$$

Ainsi on distingue deux cas, soit on est arrivé à un noeud sortie ( $b_j < 0$ ), soit  $b_j \geq 0$  et alors d'après la loi des noeuds :

$$\sum_{(j,k) \in E} x_{jk} > 0$$

Donc il existe  $k$  tel que  $x_{jk} > 0$ . Ainsi, comme  $G$  n'a pas de cycle et est fini, en itérant ce principe un nombre fini de fois, on a construit un chemin de  $i$  vers un noeud sortie qui est inclus dans le support de  $x$ . De la même façon, on peut construire un chemin d'un noeud entrée vers le noeud  $i$  inclus dans le support de  $x$ . On a alors un **chemin reliant un noeud entrée noté  $i'$  à un noeud sortie noté  $j'$  qui est inclus dans le support de  $x$** . On note  $\gamma$  ce chemin. Définissons  $y$  le flot valant 1 sur ce chemin et 0 sur le reste du graphe et  $\alpha = \min_{(i,j) \in \gamma} y_{ij} > 0$ . Alors  $x' = x - y\alpha$  est un flot, admissible par définition de  $\alpha$  dans le graphe  $G'$  obtenu à partir de  $G$  en changeant  $b_{i'}$  et  $b_{j'}$  en  $b_{i'} - \alpha$  et  $b_{j'} + \alpha$  respectivement. Par ailleurs, le support de  $x'$  est strictement inclus dans celui de  $x$ . Donc en itérant ce principe un nombre fini de fois (car les inclusions de supports sont strictes), le flot deviendra nul et donc la décomposition sera obtenue.  $\square$

Nous justifions maintenant que l'approche par PLNE présentée en 2.1.4 est applicable au problème de flot à coût minimum où les flots exogènes, les contraintes et les capacités sont énitiers comme cela sera le cas dans notre modèle. Dans un premier temps nous cherchons à réécrire (7) pour retrouver la forme standard de (2). Pour cela nous introduisons la *matrice d'incidence nœuds-arcs*  $A \in \mathbb{R}^{m \times n}$  du graphe  $G$  qui permet une réécriture de la loi des noeuds de Kirchhoff (4) avec  $Ax = b$  :

$$A_{i,(j,k)} = \begin{cases} -1 & \text{si } i = k \\ 1 & \text{si } i = j \\ 0 & \text{sinon} \end{cases}$$

où la  $i$ -ème ligne de  $A$  se rapporte au sommet  $i$  du graphe, et la colonne notée  $(j, k)$  à l'arc  $(j, k)$  du graphe (avec un coefficient 1 à la ligne  $j$  pour le noeud entrant et un coefficient  $-1$  à la ligne  $k$  pour le noeud sortant). On peut alors réécrire (7) :

$$\min_{x \in \mathbb{R}^n, Ax = b, 0 \leq x \leq u} \sum_{(i,j) \in E} c_{ij} x_{ij} \quad (8)$$

Enfin en introduisant une variable d'écart positive  $\lambda \in \mathbb{R}^n$ , les contraintes de capacités sont équivalentes à  $x + \lambda = u$  et on peut écrire la forme standard du programme linéaire (8) :

$$\min_{\tilde{x} \in \mathbb{R}^{2n}, \tilde{A}\tilde{x} = \tilde{b}, \tilde{x} \geq 0} \sum_{(i,j) \in E} c_{ij} x_{ij} \quad (9)$$

avec :

$$\tilde{x} = \begin{pmatrix} x \\ \lambda \end{pmatrix}, \quad \tilde{b} = \begin{pmatrix} b \\ u \end{pmatrix}, \quad \tilde{A} = \begin{pmatrix} A & 0 \\ I_n & I_n \end{pmatrix}$$

Finalement nous cherchons à résoudre le PLNE suivant :

$$\min_{\tilde{x} \in \mathbb{Z}^{2n}, \tilde{A}\tilde{x} = \tilde{b}, \tilde{x} \geq 0} \sum_{(i,j) \in E} c_{ij} x_{ij} \quad (10)$$

Montrons alors que ce problème de flot à coût minimum entre dans le cadre des hypothèses de (2.2) :

**Lemme 2.4.** *La matrice d'incidence nœuds-arcs d'un graphe est totalement unimodulaire.*

*Démonstration.* Chaque arc possède un unique noeud de départ et d'arrivée. La matrice  $A$  a donc par définition un unique 1 et un unique  $-1$  par colonne. Le lemme de Poincaré (2.3) permet de conclure.  $\square$

Pour la proposition suivante on suppose que l'ensemble des flots admissibles est borné et non-vide. Dans ce cas on peut théoriquement énumérer toutes les solutions admissibles entières et le problème (10) admettra un solution optimale sur cet ensemble fini. La proposition (2.1) assure alors l'existence d'une solution optimale basique.

**Proposition 2.4.** *On suppose que les capacités et les flots exogènes sont entiers :  $b_i \in \mathbb{Z}$  et  $u_{ij} \in \mathbb{N} \cup \{+\infty\}$ . Alors les sommets de l'ensemble des solutions admissibles du problème (9), appelé problème relaxé du PLNE (10), sont entiers. Si de plus l'ensemble des flots admissibles est borné et non vide, le problème (9) admet une solution optimale entière et l'algorithme du simplexe renvoie une solution optimale à valeur dans  $\mathbb{N}$ .*

Ainsi on obtient que, dans le cadre des hypothèses énoncées, résoudre (9) suffit pour résoudre (10). Or le problème relaxé est plus simple et traitable par l'algorithme du simplexe.

*Démonstration.* Montrons premièrement qu'un sommet du polyèdre des flots admissibles est à composantes entières. Rappelons qu'un sommet  $\tilde{x}$  (qui est une solution basique) vérifie  $\tilde{A}\tilde{x} = \tilde{b}$ . On a donc aussi  $Ax = b$  et  $\lambda = u - x$ . Or  $A$  est totalement unimodulaire et  $b$  entière donc par la proposition 2.2,  $x$  est entière. Puisque  $u$  est entier par hypothèse,  $\lambda$  aussi et donc  $\tilde{x}$  aussi. Montrons maintenant l'existence d'une solution optimale. Si l'ensemble des flots admissibles est borné et non-vide, alors la fonction objectif (ou fonction de coût)  $c \cdot x$  est bornée sur l'ensemble des flots admissibles et le lemme 2.1 assure l'existence d'une solution optimale au problème (9). Alors le théorème 2.1 montre qu'il admet un solution optimale basique (donc un sommet), qui est entière par ce qui précède. Ainsi (9) et (10) sont égaux et l'algorithme du simplexe qui explore les sommets, ici à valeurs dans  $\mathbb{N}^n$ , renvoie bien une solution à composantes entières.  $\square$

La proposition précédente justifie donc l'utilisation de l'algorithme du simplexe pour résoudre le problème de flot à coût minimum associé à notre problème que nous détaillons dans la section suivante. De plus la proposition 2.3 justifie la pertinence du modèle pour notre problème. En effet, pour un flot représentant une évacuation, chaque flot unitaire de cette décomposition représente un chemin d'un individu d'une entrée vers une sortie de notre stade.

## 2.3 NOTRE MODÈLE DE STADE

---

Nous souhaitons modéliser le stade à l'aide d'un graphe sur lequel appliquer un algorithme de coût minimum, qui nous donnerait le flot correspondant à un flot optimal, c'est-à-dire représentant l'évacuation la plus rapide. Nous détaillons ici la construction et la pertinence de notre modèle. Afin d'avoir un stade réel sur lequel s'appuyer dans le projet, nous nous sommes intéressés au stade Auguste Delaune à Reims que nous avons visité avant et pendant une rencontre sportive.

### 2.3.1 • GRAPHE STATIQUE

Dans un premier temps, on représente le stade par un graphe statique  $G_s$  à partir de la structure du stade. Il est un intermédiaire avant le graphe dynamique que l'on détaillera ensuite, qui inclura la dimension temporelle et auquel on appliquera l'algorithme de coût minimum. Les noeuds et les arcs du graphe statique possèdent quelques attributs supplémentaires par rapport à la définition en section (2.2).

Les noeuds, dont l'ensemble est noté  $V_s$ , représentent les blocs des tribunes (là où les personnes sont assises), les halls, les embranchements ou les sorties, c'est-à-dire les endroits de stockage ou de fortes rencontres. Le sous-ensemble des sorties est noté  $S \subset V_s$ . Les arcs, dont l'ensemble est noté  $E_s$ , représentent les chemins élémentaires entre les noeuds, c'est-à-dire les allées et les escaliers. Il est évident que dans ce graphe représentant un stade, de tout noeud il existe un chemin vers une sortie au moins.

Afin d'ajouter la dimension temporelle dans notre modèle, le temps est discrétisé selon une période de temps  $T$  que l'on choisira de manière optimale ensuite.

La capacité  $u_{ij} \in \mathbb{R}_+ \cup \{+\infty\}$  d'un arc représente la capacité dynamique du chemin, c'est-à-dire le nombre de personnes maximal qui peuvent y entrer par pas de temps  $T$ . Chaque arc est également muni d'un temps  $t_{ij} \in \mathbb{N}^*$  représentant le temps de traversée d'un individu (en nombre de pas de temps). On ajoute à chaque noeud  $i$  une capacité  $M_i$  représentant la capacité de personnes pouvant être stockées dedans.

Lors de notre visite au stade Auguste Delaune que nous détaillons plus loin, les capacités des noeuds ont été déterminées de différentes manières : le nombre de sièges pour un bloc, ou à partir de la surface de la zone (en appliquant la norme de trois personnes au maximum par mètre carré imposée par l'article PA 2 de l'arrêté du 6 janvier 1983). Les capacités dynamiques ont été obtenues à partir de la largeur des allées, en utilisant une formule empirique de Pauls [5]. Cette formule s'écrit :  $f(x) = 0.206w(\frac{x}{w})^{0.27}$ , où  $f(x)$  est le flux (en nombre de personnes par seconde) pour  $x$  personnes qui veulent emprunter un couloir de largeur effective  $w$  (largeur réelle à laquelle on retranche 0.3 mètre). Ainsi pour un arc  $(i, j)$ , on a  $u_{ij} = T f(x)$ . Cette formule, que l'on retrouve dans la littérature associée à la recherche opérationnelle appliquée aux problèmes d'évacuation [5], donne des résultats cohérents en fixant  $x$  comme un majorant du nombre total de personnes qui emprunteraient l'arc, sachant que  $f$  varie peu avec les variations de  $x$ . Il est évident que cette formule peut être améliorée, notamment avec une éventuelle approche par la mécanique des fluides, mais nous nous sommes intéressés à d'autres sujets.

Le temps de traversée d'un arc a été mesuré pour une personne se déplaçant à allure normale.

Ces deux valeurs, la capacité dynamique et le temps de traversée, sont considérées indépendantes de la densité de personnes et des individus. Cependant cela est en partie dû au fait que les individus peuvent parfois rester bloqués avant de se redéplacer à vitesse normale, là où en réalité leur vitesse de déplacement est lissée.

### 2.3.2 • GRAPHE DYNAMIQUE

À partir de ce graphe statique, nous souhaitons créer un graphe dynamique, intégrant la dimension temporelle, tel que tout flot admissible entier correspond à un plan d'évacuation et réciproquement, grâce à la proposition 2.3. On utilise alors la procédure de Ford et Fulkerson qui est la suivante.

On rappelle que le temps est discrétisé selon un pas de temps  $T$ , et on note  $N_{max}$  le nombre de périodes de temps auxquelles on s'intéresse. On choisit  $N_{max}$  fini et assez grand pour avoir

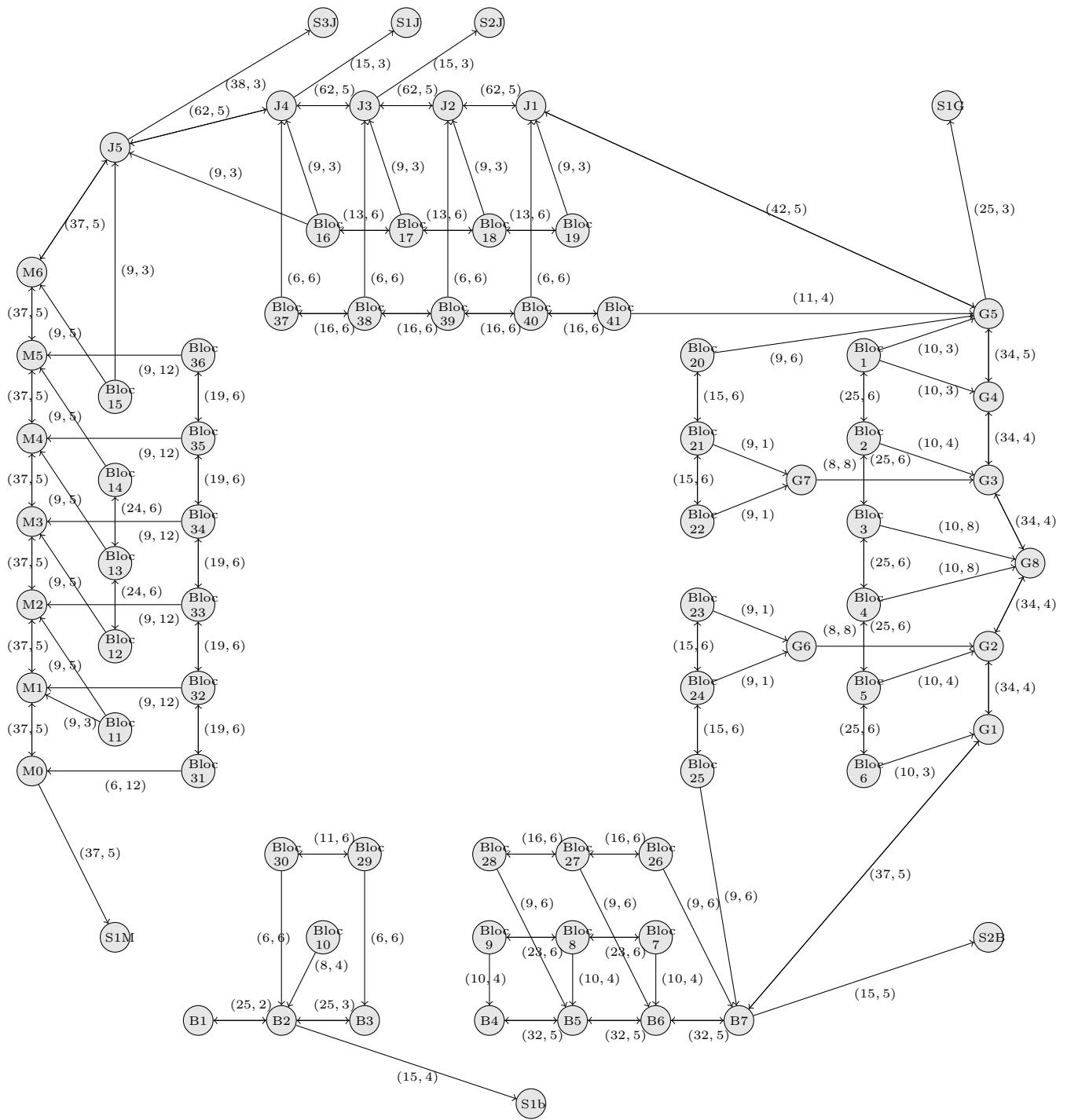


FIGURE 4 – Graphe statique du stade Auguste Delaune

une évacuation complète possible.

Notre graphe de flot devra prendre en compte les caractéristiques suivantes :

- i) Étant dans un noeud à un instant  $t$ , un individu peut soit emprunter un arc  $(i, j)$  pour aller dans un autre noeud pour y arriver à l'instant  $t + t_{ij}$ , soit rester dans ce noeud au temps  $t + 1$ .
- ii) Au maximum  $u_{ij}$  personnes peuvent entrer dans l'arc  $(i, j)$  à un instant donné.
- iii) Au maximum  $M_i$  personnes peuvent se trouver dans le noeud  $i$  à un instant donné.

Voici comment on construit le graphe dynamique à partir du graphe statique pour vérifier ces conditions, cette construction étant illustrée figure 4 :

- Chaque noeud  $i$  de  $V_s$  est dupliqué  $N_{max}+1$  fois selon un indice de temps  $t$  pour représenter le noeud à chaque instant et noté  $i_t$ .
- Afin d'avoir iii), chaque nouveau noeud est dupliqué en deux noeuds, notés  $i_t^1$  et  $i_t^2$ , et reliés par un arc  $(i_t^1, i_t^2)$  de capacité  $M_i$ . Les individus quitteront le noeud par le noeud 2 et y arriveront par le noeud 1.
- On ajoute les arcs en deux étapes. Pour chaque noeud  $i$  de  $V_s$ , on ajoute les arcs  $(i_t^2, i_{t+1}^1)$  pour tout  $t \in \{0, \dots, N_{max} - 1\}$  de capacité  $M_i$  pour modéliser le fait de rester dans le noeud. Ensuite, pour chaque arc  $(i, j)$  de  $E_s$ , on ajoute les arcs  $(i_t^2, j_{t+t_{ij}}^1)$  pour tout  $t \in \{0, \dots, N_{max} - t_{ij}\}$  pour modéliser le déplacement d'un noeud  $i$  à l'instant  $t$  vers un noeud  $j$  en prenant en compte le temps de traversée.
- Enfin, dans notre modèle, on crée un noeud *Sortie*, et on crée des arcs  $(s_t^2, Sortie)$  pour tout  $s \in S$  et  $t \in \{0, \dots, N_{max}\}$  de capacité infinie, notre objectif étant de minimiser le temps d'évacuation global sans se soucier de quelle sortie les individus prennent.

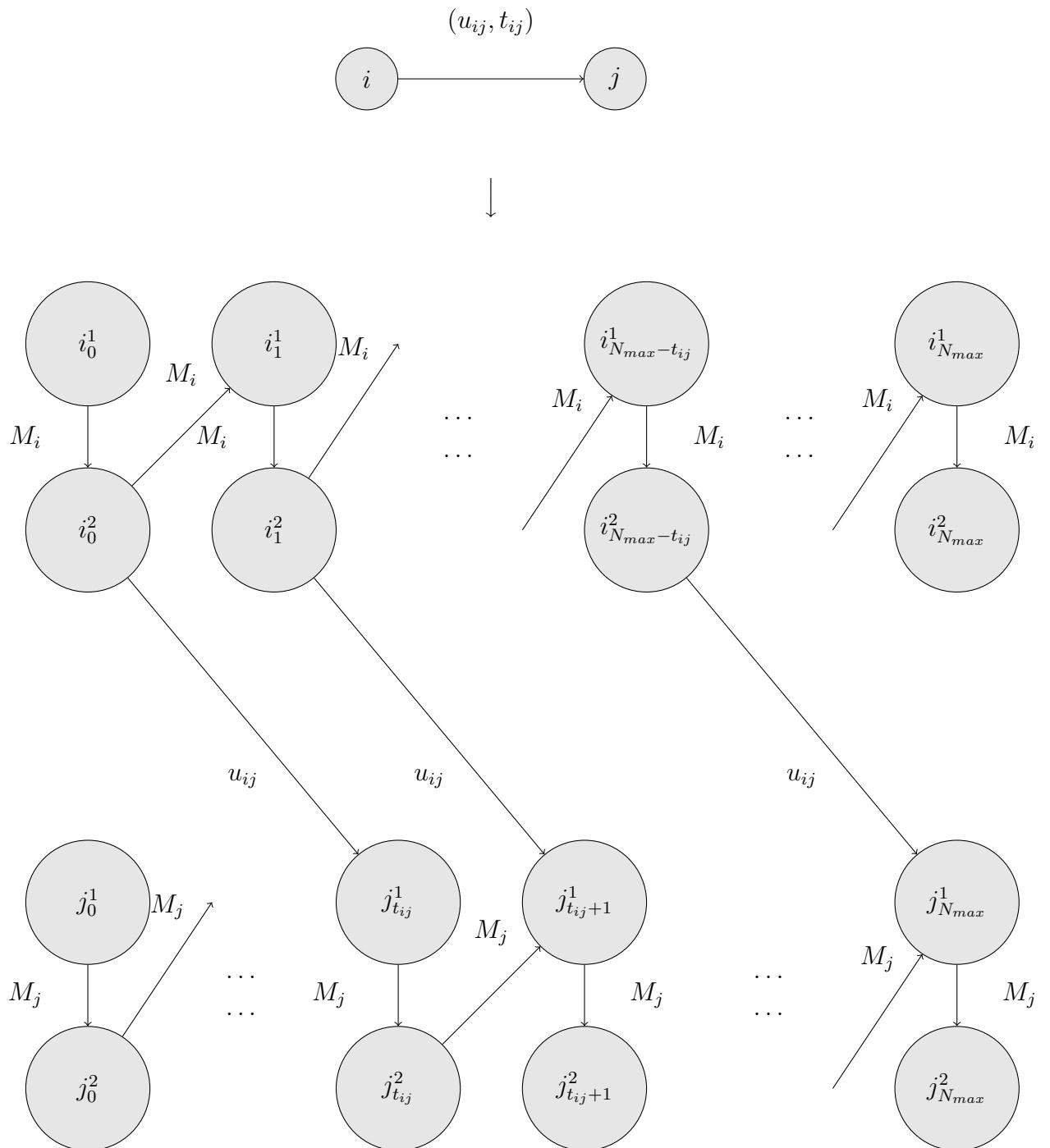


FIGURE 5 – Illustration de la construction du graphe dynamique à partir du graphe statique

Ainsi, pour résumer :

$$G = (V, E)$$

où  $V = V_s \times \{0, \dots, N_{max}\} \times \{1, 2\} \cup \{\text{Sortie}\}$  et  $E = E_1 \cup E_2 \cup E_3 \cup E_4$

$$\begin{aligned} \text{avec } E_1 &= \left\{ (i_t^2, j_{t+t_{ij}}^1), ((i, j), t) \in E_s \times \{0, \dots, N_{max} - t_{ij}\} \right\}, \\ E_2 &= \left\{ (i_t^2, i_{t+1}^1), (i, t) \in V_s \times \{0, \dots, N_{max} - 1\} \right\}, \\ E_3 &= \left\{ (i_t^1, i_t^2), (i, t) \in V_s \times \{0, \dots, N_{max}\} \right\}, \\ E_4 &= \left\{ (s_t^2, \text{Sortie}), (s, t) \in S \times \{0, \dots, N_{max}\} \right\}. \end{aligned}$$

Enfin, il reste à ajouter les flots entrants exogènes. Afin de représenter les personnes dans les arcs au début de l'évacuation, pour chaque noeud  $i$  du graphe statique, on ajoute un flot entrant exogène  $b_{i_0^1} \geq 0$  qui correspond au nombre de personnes dans le noeud au départ. On ajoute au noeud *Sortie* un flot sortant de valeur correspondant à la totalité des personnes dans l'enceinte au départ ( $b_{\text{Sortie}} = - \sum_{i \in V} b_{i_0^1}$ ).

Ce graphe possède donc des capacités et des flots exogènes entiers. De plus, il ne contient pas de circuit, n'ayant pas de retour en arrière dans le temps.

Il est facile de vérifier que si le graphe statique contenait  $n$  sommets, dont  $s$  sorties et  $a$  arcs, le graphe dynamique possède au plus  $2n(N_{max} + 1) + 1$  sommets et  $(2n + a + s)N_{max} + n + s$  arcs.

Ceci est une borne supérieure car il est possible de réduire ce nombre d'arcs et de noeuds de manière assez importante en supprimant de nombreux noeuds et arcs inutiles en étudiant de près le graphe. Il est par exemple possible de supprimer de nombreux arcs et arcs proches des sorties aux premiers temps et d'autres proches des sources aux temps longs.

Ainsi, le choix de la période de temps  $T$  doit satisfaire un compromis entre deux choses. La première est la limitation de la taille du graphe et donc de la complexité de la résolution. En effet, la taille du graphe dépend de  $N_{max}$ , qui est déterminé par la relation  $N_{max} = \frac{t_{evac}}{T}$  où  $t_{evac}$  est le temps d'intérêt de l'évacuation, qu'on prend légèrement supérieur au temps d'évacuation attendu, aux alentours de 10 ou 15 minutes. Le choix d'un temps  $T$  trop petit engendre une taille du graphe trop importante. Le deuxième raison est de choisir un temps  $T$  qui divise tous les temps de trajet des arcs, afin de satisfaire notre discréétisation du temps. Dans l'idéal,  $T$  devrait être pris assez petit pour bien prendre en compte les différences de temps de trajet dans chaque arc.

Nous avons donc choisi  $T$  comme le plus grand diviseur commun du temps de trajet de tous les arcs, afin qu'il les divise tous. Nous avons parfois légèrement modifié les temps de traversée de quelques secondes afin d'avoir un PGCD plus grand que un. Dans la majorité de nos simulations,  $T$  vaut environ 3 secondes, ce qui permet une résolution assez précise et rapide.

Finalement le graphe dynamique ainsi construit respecte les hypothèses relatives au graphe énoncées dans la proposition 2.2. Intéressons nous maintenant à la définition de la fonction objectif.

### 2.3.3 • CHOIX DU COÛT ET RÉSOLUTION

Le graphe étant construit, il reste à définir le coût avant d'appliquer l'algorithme du simplexe pour obtenir un flot admissible qui le minimise. Le coût doit être ainsi choisi de manière à ce que ce flot soit celui qui correspond à une évacuation de temps minimal, c'est-à-dire de telle sorte que le dernier individu sorte du stade le plus rapidement possible.

Il est donc défini de la manière suivante. On note  $n$  le nombre de personnes à évacuer dans le stade Auguste Delaune (qui correspond au flot sortant du sommet *Sortie*, qui est d'environ 21000 au maximum). À chaque arc reliant un noeud de sortie  $s$  au noeud *Sortie* à l'instant  $t$ , on attribue un coût  $c_{s^2_t - \text{Sortie}} = (n+1)^t + 1$ . Les autres arcs ont un coût de 1.

Nous n'avions pas ajouté ce coût de 1 aux autres arcs au départ, car cela n'était pas nécessaire pour obtenir un flot optimal. Cependant, plusieurs solutions étaient possibles, par exemple certains individus faisaient des aller-retours entre les sommets plutôt que d'y patienter, les coûts des flots correspondant aux deux situations étant identiques. L'ajout de 1 à chaque coût permet d'éviter les déplacements inutiles. Cette modification de la fonction de coût devient naturelle lorsqu'on visualise ce problème de programmation linéaire à l'aide d'un polyèdre comme dans la première partie, ce que l'on détaille par la suite.

Comme les capacités et les flots exogènes sont entiers, on sait qu'il existe un flot admissible entier  $x^*$  de coût minimum (si  $N_{max}$  est assez élevé). Ce flot correspond à une évacuation de temps minimal.

*Démonstration.* On ne prend pas en compte le coût unitaire pour chaque arc dans la démonstration car elle serait similaire. On note  $\mathcal{T}_{min}$  le temps d'évacuation optimal, et  $\mathcal{T}^*$  le temps d'évacuation correspondant au flot  $x^*$  de coût minimal (en nombre de pas de temps) :  $\mathcal{T}^* = \max \{t \in \{0, \dots, t_{max}\} : \exists s \in S, x_{s^2_t - \text{Sortie}}^* > 0\}$ .

On a déjà par définition, de  $\mathcal{T}_{min} : \mathcal{T}_{min} \leq \mathcal{T}^*$ .

Soit  $x$  un flot admissible entier se terminant en temps  $\mathcal{T}_{min}$ . Son coût s'écrit :

$$\begin{aligned} c(x) &= \sum_{s \in S} \sum_{t=0}^{\mathcal{T}_{min}} x_{s^2_t - \text{Sortie}} (n+1)^t \leq (n+1)^{\mathcal{T}_{min}} \sum_{s \in S} \sum_{t=0}^{\mathcal{T}_{min}} x_{s^2_t - \text{Sortie}} \\ &= (n+1)^{\mathcal{T}_{min}} n \end{aligned}$$

Le coût pour  $x^*$  s'écrit, sachant qu'il existe  $s \in S$  tel que  $x_{s^2_{\mathcal{T}^*} - \text{Sortie}}^* \geq 1$  :

$$c(x^*) = \sum_{s \in S} \sum_{t=0}^{\mathcal{T}^*} x_{s^2_t - \text{Sortie}} (n+1)^t \geq (n+1)^{\mathcal{T}^*}$$

On alors, si  $\mathcal{T}_{min} < \mathcal{T}^*$ ,  $c(x^*) > c(x)$  car  $\mathcal{T}_{min}$  et  $\mathcal{T}^*$  sont entiers, ce qui est absurde.  
On a donc  $\mathcal{T}_{min} = \mathcal{T}^*$ .

□

Revenons sur la modification de la fonction de coût (celle où les arcs qui avaient un coût nul ont un coût de 1) à travers un exemple qui nous donnera une intuition sur cette réduction de l'ensemble des solutions. Prenons le cas d'une évacuation d'un bâtiment qui ne possède que deux pièces.

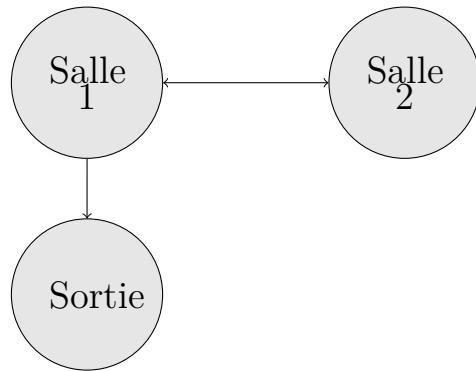


FIGURE 6 – Graphe statique de l'exemple

Initialement 11 personnes sont situées dans la salle 1. À chaque pas de temps, 5 personnes peuvent sortir. Il n'y a pas de contrainte sur l'arête reliant la salle 1 à la salle 2. Nous nous intéressons aux différentes évacuations possibles qui ne durent que 3 pas de temps (temps minimal).

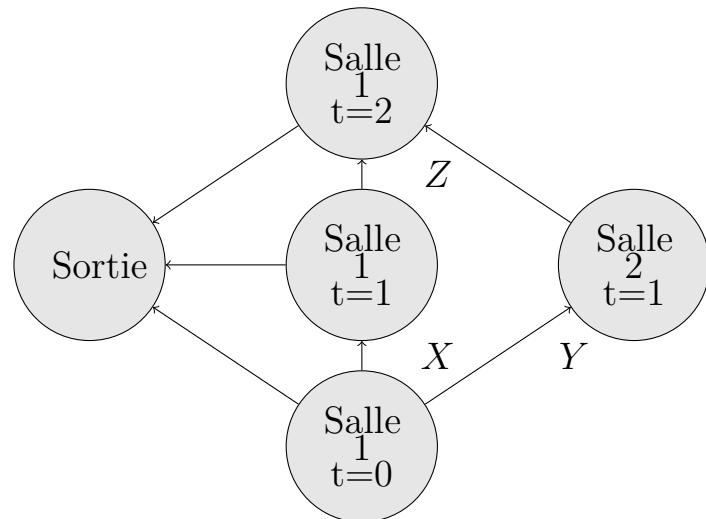


FIGURE 7 – Graphe dynamique de l'exemple

Le fait que le flot est admissible nous permet de réduire le problème d'optimisation à 3 variables ( $X, Y, Z$ ). Les contraintes sur les flots nous donnent le système suivant :

$$\begin{cases} X + Y \geq 6 \\ X - Z \leq 5 \\ Z + Y \leq 5 \\ X, Y, Z \geq 0 \end{cases}$$

On obtient donc le polyèdre de la figure 8 :

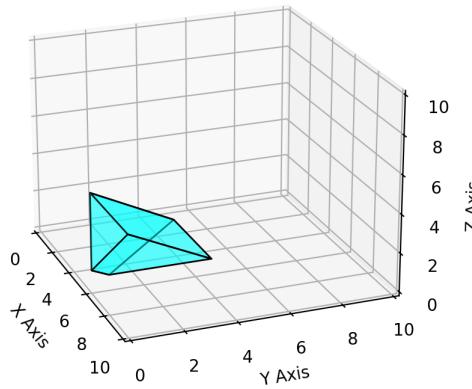


FIGURE 8 – Espace des flots admissibles.

Sans l'ajout des coûts sur les arêtes, on obtient la fonction de coût suivante :

$$f(X, Y, Z) = (11 - X - Y)e^0 + (X - Z)e^1 + (Z + Y)e^2$$

Sur le polyèdre le coût minimal est de  $C = 5e^0 + 5e^1 + 1e^2$ . Cependant l'intersection de l'hyperplan  $C = (11 - X - Y)e^0 + (X - Z)e^1 + (Z + Y)e^2$  avec le polyèdre est une arête, il y a donc plusieurs solutions optimales.

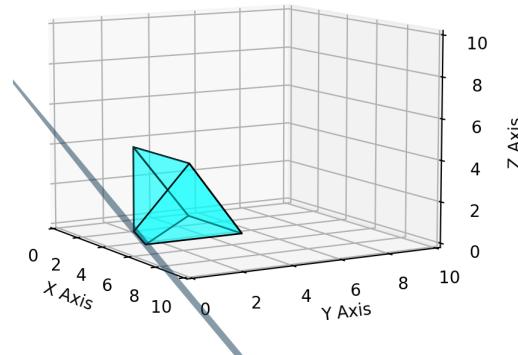


FIGURE 9 – Ensemble des flots optimaux pour  $f$ .

Cependant en ajoutant le coût sur l'arête  $salle1 - salle2$ ,  $g(X, Y, Z) = (11 - X - Y)e^0 + (X - Z)e^1 + (Z + Y)e^2 + 2Y$ , l'intersection de l'hyperplan  $g(X, Y, Z) = C$  est un sommet, il y a donc une unique solution.

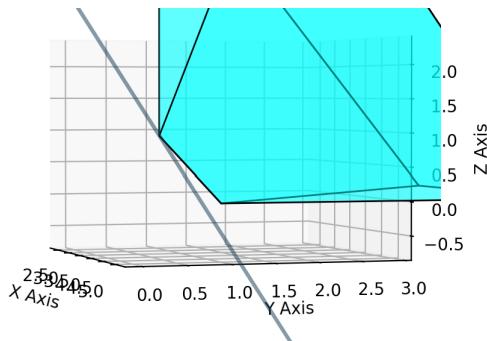


FIGURE 10 – Ensemble des flots optimals pour  $g$ .

C'est assez logique : comme pour le stade, les personnes qui attendent deux pas de temps dans une salle peuvent aussi faire le choix de faire un aller-retour dans une salle vide. Cela crée plusieurs solutions optimales. Ces nouveaux coûts permettent de pénaliser ces comportements, réduisant ainsi le nombre de solutions optimales.

### 3

## NOTRE DÉMARCHE AVEC LE STADE AUGUSTE DELAUNE

### 3.1 OBSERVATIONS ET PRISES DE MESURES

Suite à nos échanges avec le responsable de la sécurité du Stade de Reims, club de ligue 1, nous avons été invités au stade Auguste Delaune (capacité de 20519 places) le jour du match Reims - Le Havre, le 20 décembre 2023. Ce fut pour nous l'occasion de confronter la théorie à la réalité du terrain. Nous avions à cette occasion deux objectifs : 1. Effectuer les mesures nécessaires pour alimenter les paramètres numériques de notre graphe de flots. 2. Observer l'évacuation du stade en temps réel pour pouvoir comparer nos futurs résultats théoriques avec des observations expérimentales. Notre venue au stade nous a aussi permis d'identifier des scénarios d'évacuation contraints intéressants à étudier, ceci grâce à nos échanges avec le personnel de sécurité sur place. Notre journée au stade s'est ainsi découpée en trois phases :

#### • VISITE DE L'INFRASTRUCTURE

Nous avons pu découvrir les différentes tribunes du stade ainsi que les accès les reliant ce qui nous a permis de mieux appréhender les espaces et les couloirs figurant sur les plans que nous avions récupérés. Cela nous a permis d'observer la configuration des accès principalement sous tension lors de l'évacuation des spectateurs. L'accès le plus sensible lors de l'évacuation se situe au sud du stade : au coin entre les tribunes Germain et Joncquet. Cet accès est matérialisé par une grille d'accès située au coin de l'esplanade en bas de la tribune Joncquet. C'est par cet accès que la majorité des spectateurs des tribunes Meano et Joncquet, ainsi qu'une partie du public de la tribune Germain, sort du stade. Le plan ci-après mentionne les capacités totales des différentes tribunes ainsi que les flux d'évacuations prévus en fonction de la jauge : environ 12000 spectateurs en jauge basse, et 20000 spectateurs en jauge maximale. Nous anticipons donc une importante densité de spectateurs sur l'esplanade à la sortie de la tribune Joncquet.

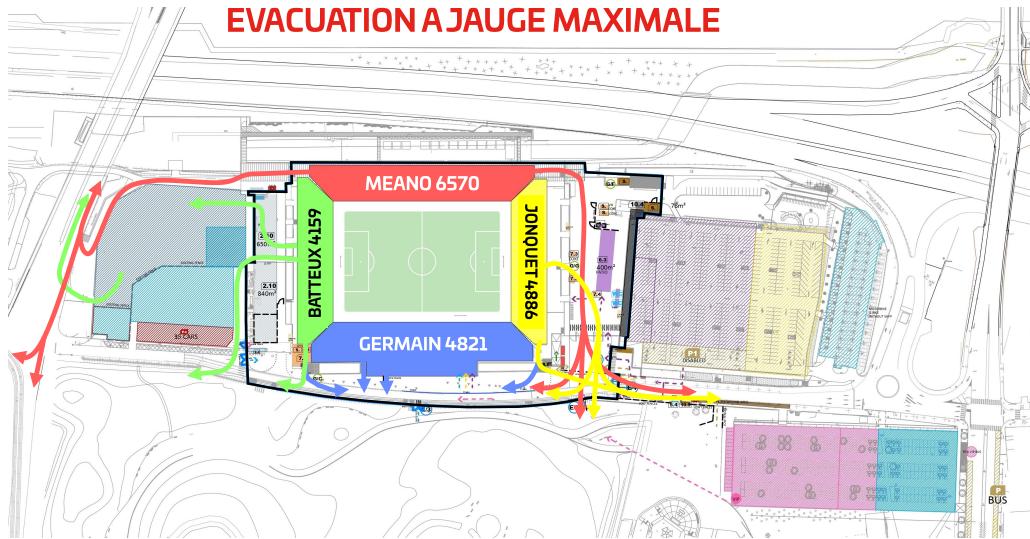


FIGURE 11 – Plan du stade et flux sortants (jauge maximale  $\approx 20500$  spectateurs)

- PRISE DE MESURES POUR AVOIR DES DONNÉES NUMÉRIQUES

Nous avons ensuite passé l'après-midi à récupérer les données manquantes pour notre modèle en effectuant des mesures de largeur des allées, des surfaces des différents espaces (coursives, couloirs, etc.), et de temps de parcours nécessaires à la traversée de chaque escalier et de chaque couloir (en essayant d'adapter l'allure à celle d'une évacuation). Ces différentes mesures nous permettent d'avoir les données nécessaires à la modélisation sur NetworkX du stade : les surfaces pour estimer les capacités de chaque noeud, les temps de trajets pour le coût en temps de chaque arc, et enfin les largeurs des allées pour calculer les capacités dynamiques des arcs (capacité maximale d'un arc par unité de temps) à l'aide de la formule empirique de Pauls.



FIGURE 12 – Esplanade devant la coursive de la tribune Joncquet et grille au coin sud.

## • OBSERVATION DE L'ÉVACUATION DU STADE POST-MATCH

Nous avons également observé avec attention l'évacuation du stade à la fin du match, en nous répartissant en trois binômes : Lucas et Matthieu au PC sécurité ayant accès aux différentes caméras de surveillance du stade permettant d'avoir une vision d'ensemble de l'évacuation ; Tanguy et Félix à l'angle Jonquet - Meano, et enfin Sophie et Arthur à l'intérieur du stade pour visualiser les issues saturées et chronométrier le temps d'évacuation totale. L'affluence lors du match était de 14835 spectateurs , dont 344 visiteurs, ce qui correspond à une jauge moyenne (stade rempli aux 2/3). Le coup de sifflet final a eu lieu à 22h53 et l'évacuation a pris 6 minutes 35 secondes pour la majorité des spectateurs. L'évacuation s'est globalement bien déroulée, il n'y a eu quasiment aucun problème et nous avons observé très peu de ralentissements. Cela est dû au fait qu'à l'approche de la fin du match, la sécurité ouvre toutes les grilles extérieures, y compris celles qui étaient fermées avant le match permettant l'évacuation anticipée d'une partie des spectateurs. Cela permet donc une grande fluidité aux abords du stade. Une autre raison de cette fluidité est le fait que les gens ne sont pas tous partis du stade au même moment, certains restant plus longtemps pour célébrer la victoire. C'est principalement le cas pour la tribune Jonquet qui comporte le Cop de supporters Remois. Les spectateurs restés pour saluer les joueurs sont ainsi restés jusqu'à 23h05. Comme attendu le principal ralentissement a eu lieu dans les escaliers de la tribune Meano, dans l'escalier 12 principalement.

De plus, nous avons pu avoir une bonne vision d'ensemble des problèmes liés à l'évacuation du stade, en nous appuyant sur nos conversations avec les responsables de la sécurité. Ils ont notamment insisté sur la nécessité de bien séparer les flux de supporters locaux de ceux de supporters adverses. Un autre point important est la sortie des supporters de la tribune Meano, celle-ci étant collée à une rivière et une voie rapide. Il n'y a que deux sorties pour la coursive Meano : au nord et au sud. Nous avons également pu observer un phénomène étonnant pour la sortie nord fermée à l'arrivée du public pour séparer locaux et adversaires : les supporters ne l'empruntent pratiquement pas non plus à la sortie, même si elle était ouverte. Et ce car chaque individu préfère emprunter le même chemin qu'à l'aller plutôt que de modifier son parcours, même si celui-ci est beaucoup plus rapide. C'est un aspect qu'il est difficile de prendre en compte informatiquement, car l'algorithme va nous dire que les supporters vont logiquement sortir par la sortie la plus rapide. Ce phénomène explique en partie les ralentissements observés dans la tribune Meano.

En outre, la grille sud-est au coin Meano-Jonquet était fermée afin d'éviter un flux incontrôlable dans le parking. Il en résulte un temps d'évacuation certes plus long, mais plus économique en ressources humaines (stadiers) et matérielles (barrières de sécurité notamment). Ces considérations rentrent évidemment dans l'équation pour les responsables de la sécurité, et représentent des limites de l'application de notre théorie.

Ainsi nous doutons que notre modélisation va se heurter à la réalité car nous tentons dans un premier temps de modéliser une évacuation d'urgence, ce qui n'était pas le cas lors de notre visite.



FIGURE 13 – Notre groupe au stade.

## 3.2 MODÉLISATION NUMÉRIQUE

---

À partir de ces données, nous avons modélisé le stade informatiquement (le code est donné en annexe), grâce à la bibliothèque NetworkX de Python, adaptée pour les graphes de flots. Dans le code, nous avons appliqué la méthode décrite en partie 2.3. Après avoir entré toutes les valeurs de notre graphe statique manuellement, nous avons appliqué la procédure afin de construire la graphe dynamique. Nous utilisons l'algorithme du simplexe de NetworkX.

Nous avons rencontré plusieurs difficultés avant d'arriver à des résultats exploitables. D'abord, la complexité et la singularité du stade ont demandé pas mal de travail pour entrer les valeurs. Le choix de  $N_{max}$  était aussi important pour obtenir une solution, mais aussi l'obtenir en un temps acceptable. Avec une période de 3 secondes et un intérêssement à au moins 200 périodes, nous obtenions le résultat en quelques minutes. Le coût obtenu était également très élevé, aux alentours de  $10^{64}$ , ce qui était permis par le type 'int' de Python qui peut s'étendre dynamiquement et accueillir des valeurs aussi grandes que nécessaires, limitées seulement par la mémoire.

L'exploitation des résultats s'est fait de plusieurs manières. D'abord avec la génération d'un document CSV pour chaque évacuation, qui représente le nombre de personnes dans chaque noeud et arc à tout instant de l'évacuation. Ensuite, pour analyser les résultats de manière plus visuelle, nous avons utilisé la bibliothèque Pygame de Python afin de visualiser le graphe au cours du temps, ainsi que la concentration de personnes dans les zones du stade (exemple figure 14).

La résolution et la visualisation rapides du problème d'évacuation optimale étaient très intéressantes pour faire de nombreuses simulations, que l'on pourrait pratiquement faire en direct. Pour évaluer différents scénarios, il nous a suffi de retirer ou ajouter certains noeuds ou arcs dans le graphe statique.

Notre modèle permet donc d'obtenir l'évacuation optimale pour de nombreuses situations. Il ne permet toutefois pas de prendre en compte certaines contraintes possibles, qui demandent des modifications du modèle, que nous n'avons pas approfondies. Par exemple, certains individus peuvent être groupés sans vouloir être séparés au cours d'une évacuation, ou certains individus peuvent préférer certaines sorties.

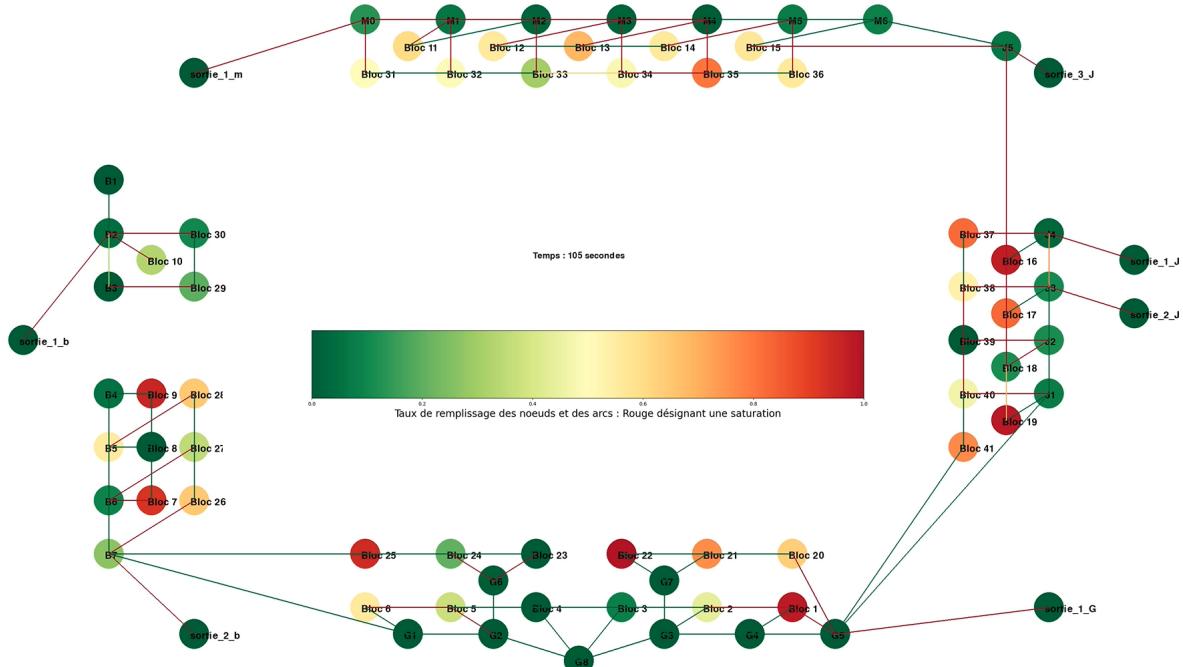


FIGURE 14 – Affichage grâce à Pygame de l'évolution des personnes dans le stade.

## 4

# ANALYSE DES RÉSULTATS ET SIMULATION DE DIFFÉRENTS SCÉNARIOS

---

Une fois le stade modélisé informatiquement, ainsi que l'algorithme de flots implémenté, nous sommes en mesure de simuler une évacuation du stade et d'en déterminer la solution optimale. Nous obtenons en sortie de notre programme un tableau Excel avec le nombre de personnes présentes à chaque instant dans chaque nœud et dans chaque arc, ce qui permet de mesurer les différents temps d'évacuation, mais aussi de voir par où elles passent. Nous obtenons également une vidéo animée qui nous permet de visualiser facilement les différents flux grâce à un code couleur. Nous sommes alors capables d'étudier différents scénarios d'évacuation, en retirant certains arcs (pour bloquer un escalier ou une sortie par exemple) ou en modifiant certains nœuds.

### 4.1 CAS GÉNÉRAL ET ÉTUDE TRIBUNE PAR TRIBUNE

---

Nous avons commencé par étudier l'évacuation dans le cas général, c'est-à-dire dans la configuration telle qu'elle est aujourd'hui, avec toutes les sorties ouvertes (y compris la sortie Méano Nord – sortie 1m), et avec un stade plein (20 519 supporters). La sortie 1B est la sortie de la tribune visiteurs ; son temps d'évacuation de 3' est donc constant dans tous les scénarios (les supporters adverses ne doivent pas croiser les locaux, et leur évacuation est donc relativement simple du point de vue du parcours). Le graphique ci-dessous (Figure 15) montre les temps d'évacuation pour chaque sortie.

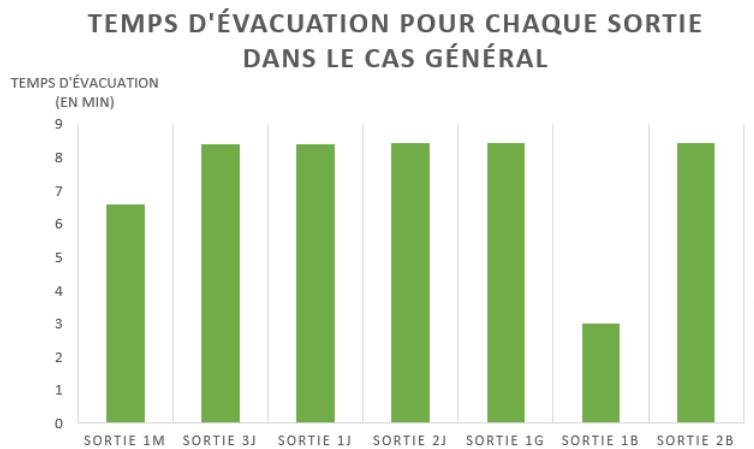


FIGURE 15 – Temps d'évacuation pour chaque sortie dans le cas général.

On voit alors que le temps d'évacuation total (pris quand la dernière personne passe la sortie) est de 8'27", ce qui est assez proche des temps observés (sachant qu'en temps normal et surtout en cas de victoire de Reims, tout le monde ne sort pas au même moment). De plus, on voit que les tribunes se vident majoritairement au même rythme, ce qui est permis par la grande interconnexion entre les différentes tribunes. Ce n'est pas forcément autant le cas dans la réalité, car cela implique aussi de dire à certaines personnes de parcourir plus de distance, mais cela pourrait donner des pistes au responsable de la sécurité pour fluidifier l'ensemble. Et en effet, si l'on ferme les connexions entre les tribunes, on obtient un temps d'évacuation bien supérieur (11'57" pour la sortie 2b de Batteux). Le graphique ci-dessous (Figure 16) le montre bien. Certaines tribunes profitent de ces fermetures et se vident plus vite toutes seules - c'est le cas de Meano (sorties 1m et 3J) notamment. En revanche, on observe que Batteux souffre fortement de ces fermetures, et la fluidité de son évacuation est donc très dépendante de son accès ou non à la tribune Germain.

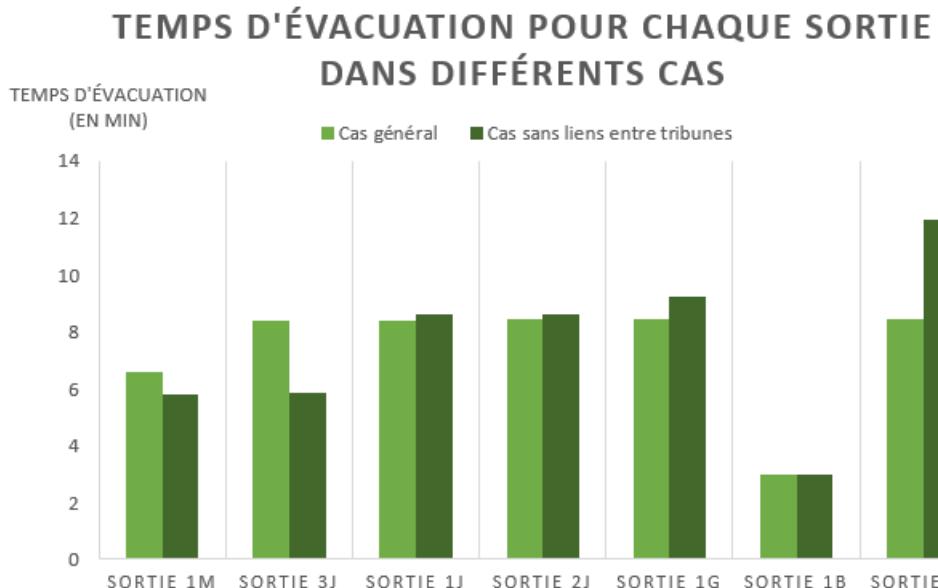


FIGURE 16 – Comparaison des temps d'évacuation pour chaque sortie dans le cas général et en coupant les liens entre tribunes.

Ces résultats montrent donc bien l'importance de l'interconnexion entre les tribunes pour une évacuation efficace.

## 4.2 INFLUENCE DE L'AFFLUENCE

Nous avons ensuite fait tourner l'algorithme pour différentes affluences : en jauge pleine (toutes les tribunes sont pleines), pour un stade rempli à deux tiers et enfin pour un stade à moitié rempli. Les résultats sont rassemblés sur le graphe ci-dessous (Figure 17). Ceux-ci

n'offrent pas de surprises : on observe que le temps d'évacuation de chaque tribune diminue bien avec la diminution de l'affluence. Cette diminution n'est pas réellement linéaire, car le temps d'évacuation tient évidemment compte du temps de trajet nécessaire à chaque spectateur, qui est indépendant d'éventuels blocages.

### TEMPS D'ÉVACUATION POUR CHAQUE SORTIE DANS LE CAS GÉNÉRAL, AVEC DES AFFLUENCES DIFFÉRENTES

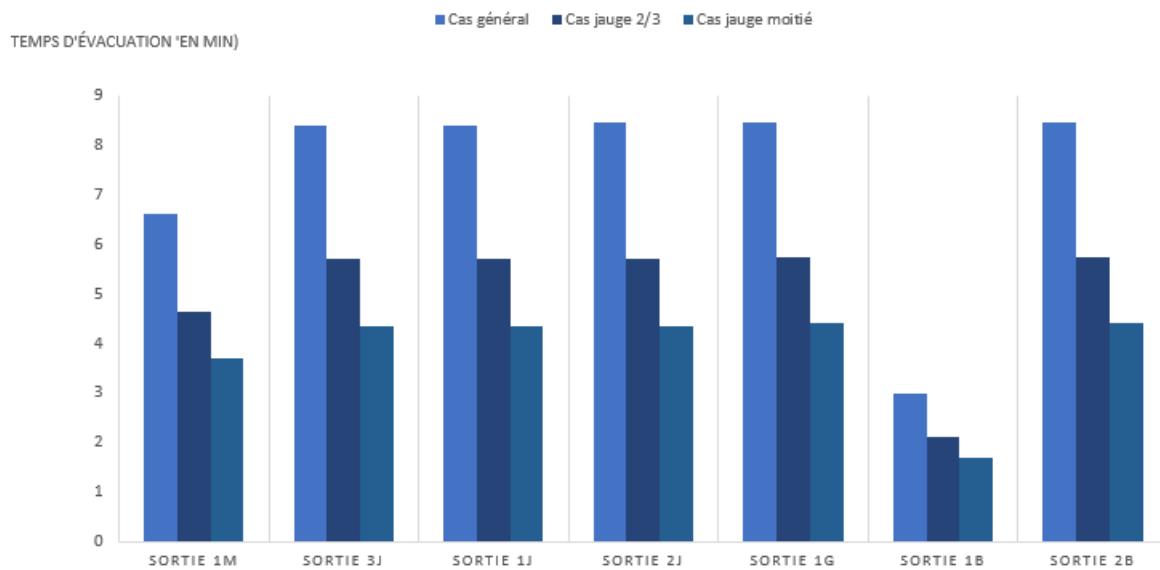


FIGURE 17 – Comparaison des temps d'évacuation pour chaque sortie pour différentes affluences.

La jauge 2/3 se rapproche très fortement de la situation que nous avons observé lors de notre déplacement sur place. En effet, l'affluence était de 14835 personnes, pour une capacité totale de 20519. Grâce à notre algorithme, nous obtenons un temps d'évacuation optimal de 5'45. Dans la réalité, nous avions observé un temps de 6'35 ; nos résultats sont donc très cohérents. D'autant plus en considérant le fait que la sortie Meano nord (1-m) est très empruntée dans notre modèle et fluidifie de manière très conséquente l'évacuation de la tribune Meano, alors que pendant l'évacuation que nous avons observée, elle n'était que peu empruntée (pour les raisons mentionnées en partie 3).

Cela nous permet tout de même d'évaluer les différents temps d'évacuation théoriques en fonction de l'affluence, ce qui est assez précieux pour le club.

## 4.3 CAS OÙ UNE SORTIE EST BLOQUÉE

Une sortie bloquée lors d'une évacuation est un problème assez complexe à gérer, puisque cela implique de rediriger tous les supporters qui sortent habituellement par celle-ci. Le blocage peut être causé par un incendie, une situation d'urgence médicale pour un individu, ou même par une intrusion terroriste à cet endroit précis. Ces situations d'urgence nécessitent de la préparation de la part de la sécurité. Nos simulations donnent un certain nombre d'indications sur la manière la plus optimale de réorienter les spectateurs. Nous ne nous intéressons pas à la tribune visiteurs dans cette étude, puisqu'elle est isolée des autres totalement (même s'il peut évidemment y avoir des cas d'urgence à son niveau aussi...).

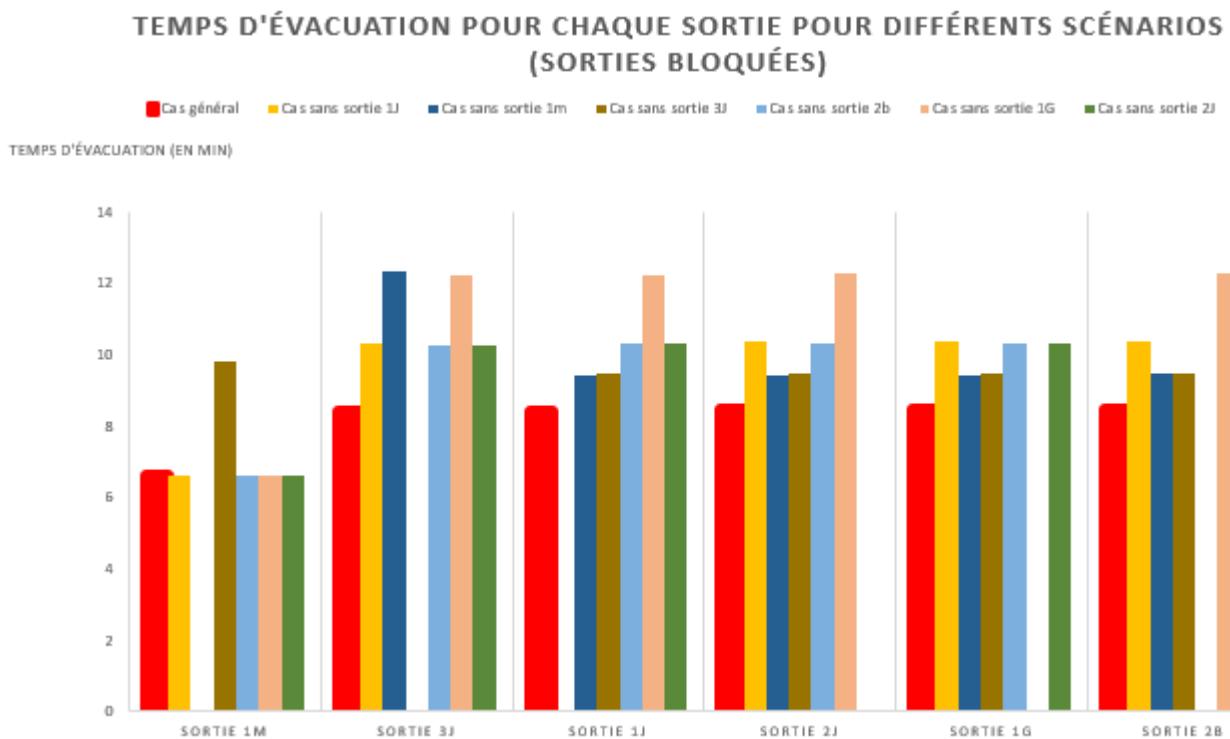


FIGURE 18 – Comparaison des temps d'évacuation pour chaque sortie avec une sortie bloquée à chaque fois.

Regardons tout d'abord l'impact de la fermeture d'une sortie sur les temps d'évacuations de toutes les autres sorties, et donc sur ces sorties en elles-mêmes. Le graphique ci-dessus (Figure 18) représente le temps d'évacuation de chaque sortie, dans le cas où une sortie est fermée à chaque fois. Le repère est évidemment le cas général, c'est-à-dire sans aucune fermeture, et est représenté en rouge. On voit ainsi que la sortie de la tribune Germain 1G est assez fondamentale dans l'évacuation du stade, puisque sa fermeture entraîne un temps d'évacuation beaucoup plus

élevé, et ce pour beaucoup d'autres sorties (en noir sur le graphique). Ce temps est de 12'18" pour les sorties 3J, 1J, 2J et 2B, contre 8'27" sans cette fermeture. Toutes les autres tribunes sont impactées par cette fermeture, et les flux se répartissent bien sur l'ensemble du stade. Mais on voit bien que cette sortie est capitale pour l'évacuation globale du stade.

Un autre point important est la fermeture de la sortie Meano nord, la sortie 1M. Elle est représentée en bleu foncé sur le graphique. Elle entraîne un report total des spectateurs de Meano vers la sortie au sud, la sortie 3J, dont le temps d'évacuation est alors de 12'21" (la configuration du stade ne permet pas d'autres reports). Cette configuration sans cette sortie est en réalité fréquemment utilisée par la sécurité du stade, comme évoqué plus haut dans le rapport, car on ne peut pas entrer par cet endroit (à cause de la tribune visiteurs). Les spectateurs ont ainsi une forte tendance à emprunter le même chemin qu'à l'aller, et ce même si cela implique de contourner tout le stade. Dès lors, même si cette sortie 1M est ouverte, elle n'est que très peu empruntée – et nous avons pu l'observer. Cela est évidemment problématique pour la durée d'évacuation de la tribune Meano, comme le montre notre étude. Car l'utilisation de tout le potentiel de cette sortie serait d'un gain considérable. Nous pouvons donc conseiller aux équipes de sécurité de tout faire pour informer de la présence de cette sortie. Le Stade de Reims compte 1/3 d'abonnés (qui viennent donc à tous les matchs) ; commencer par eux pourrait être une solution (la forme reste à déterminer). Ils permettraient ensuite de faire voir la sortie aux autres à chaque match, puisqu'elle serait déjà largement empruntée.

La fermeture des autres sorties - 1J, 2J et 2b- a un également un impact non négligeable sur la durée d'évacuation, qui passe à plus de 10 minutes. Mais on voit bien sur le graphique que les spectateurs sont bien redistribués sur les autres sorties, ce qui permet une évacuation qui est toujours relativement efficace. Ces trois sorties ont des configurations assez similaires, ce qui explique la ressemblance de ces situations. De plus, le blocage de la sortie 3J donne un temps d'évacuation un peu plus court que les trois précédentes, et celui-ci impacte principalement la tribune Meano, le reste du stade étant peu affecté.

Ainsi, cette étude permet de visualiser de manière assez précise l'impact de la fermeture de chacune des sorties sur l'évacuation de tout le stade.

#### 4.4 CAS OÙ UN ESCALIER EST BLOQUÉ

---

Une autre situation intéressante à traiter est le cas où un escalier est bloqué. Cette situation peut être assez fréquente, car il suffit que quelqu'un se blesse dedans, ou qu'il soit bloqué par un départ d'incendie par exemple, pour le rendre inutilisable. Comme pour le cas des sorties bloquées, il faut alors réorienter les personnes l'empruntant en temps normal pour qu'elles puissent sortir tout de même. Nous avons donc fait différents tests avec à chaque fois un escalier bloqué. Nos résultats sont compilés dans le graphique (Figure 19) ci-dessous.

### TEMPS D'ÉVACUATION POUR CHAQUE SORTIE POUR DIFFÉRENTS SCÉNARIOS (ESCALIERS BLOQUÉS)

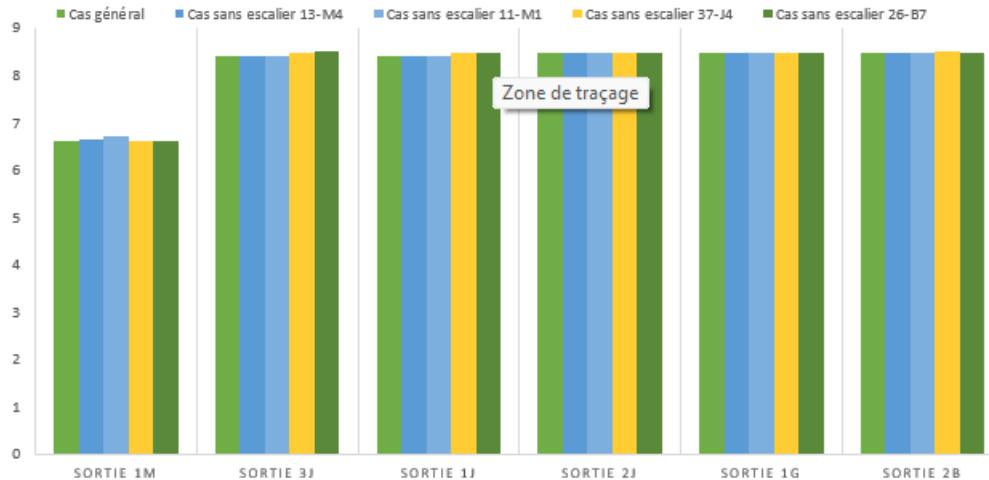


FIGURE 19 – Comparaison des temps d'évacuation avec un escalier bloqué à chaque fois.

Nous avons commencé par les escaliers les plus empruntés (le 13-M4, le 11-M1, le 37-J4, et le 26-B7). On remarque que les temps d'évacuation au niveau de chaque sortie ne varient pratiquement pas par rapport au cas général. Nous avons effectué un certain nombre de tests sur les autres escaliers, mais les résultats sont aussi similaires, nous ne les faisons donc pas apparaître ici. Nous n'avons pas identifié d'escalier capital pour l'évacuation. Celle-ci n'est pas donc dépendante d'un seul escalier en particulier, ce qui est très bénéfique pour la sécurité, particulièrement en cas d'urgence.

Cela est permis par la bonne connexion entre les différents blocs, ce qui fait que les supporters d'un bloc peuvent facilement se déverser sur les blocs voisins si leur vomitoire est bloqué. Pour mettre en valeur cela, nous avons effectué un test avec les liens entre les différents blocs coupés, obligeant les spectateurs d'un bloc à sortir par leur escalier dédié. Les résultats sont affichés sur le graphe ci-dessous (Figure 20).

**TEMPS D'ÉVACUATION POUR CHAQUE SORTIE DANS LE CAS GÉNÉRAL  
ET DANS LE CAS OÙ LES BLOCS NE SONT PAS RELIÉS ENTRE EUX**

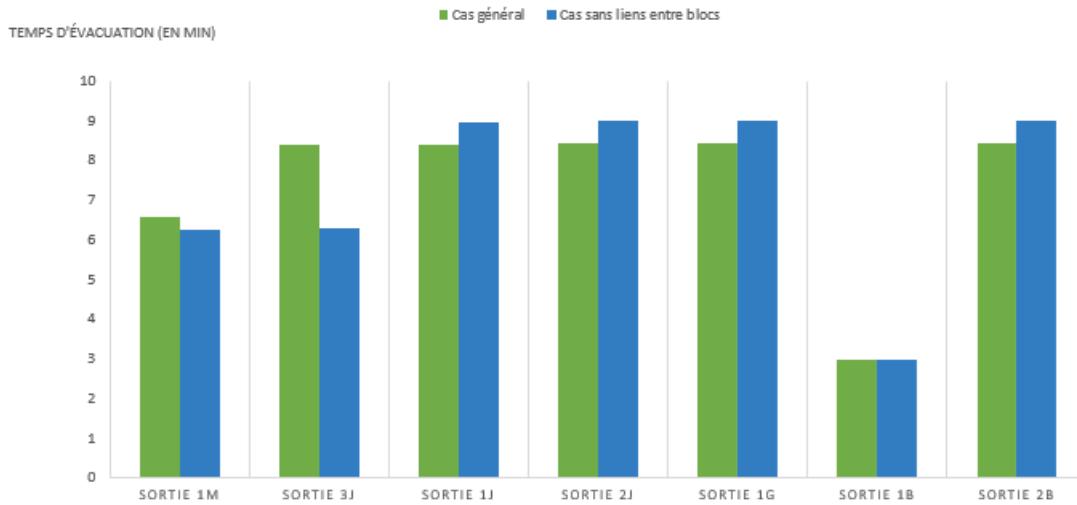


FIGURE 20 – Comparaison des temps d'évacuation entre le cas général et le cas où les différents blocs ne sont pas reliés.

Comme dans le cas où les arcs entre les tribunes étaient fermés, on voit que l'interconnexion entre les blocs permet de réduire le temps d'évacuation générale (même si cela fluidifie certaines sorties). C'est le cas au niveau des tribunes Germain, Batteux et Jonquet. Seule la tribune Meano est évacuée plus rapidement sans liens entre blocs, probablement car sa structure avec des grands escaliers très réguliers permet une bonne fluidité pour chaque bloc.

Tout cela nous montre bien encore une fois que les solutions optimales d'évacuation obtenues profitent fortement de l'interconnexion entre blocs et tribunes. Cela donne aux organisateurs des informations précieuses pour leur gestion des flux.

## 5 CONCLUSION

En conclusion, ce projet a été une opportunité enrichissante pour explorer un domaine fondamental de la recherche opérationnelle en l'appliquant à un cas concret d'évacuation de stade. En utilisant la théorie des problèmes de flots dans un graphe et les méthodes de résolution par programmation linéaire en nombres entiers présentées dans la section 2, nous avons réussi à modéliser efficacement un stade de football et son processus d'évacuation. Comme le veut toute approche d'optimisation en mathématiques appliquées, notre modèle a été alimenté par des données numériques récoltées sur le terrain lors de notre visite à Reims. L'implémentation du graphe dynamique associé au Stade Auguste Delaune sur NetworkX a produit des résultats numériques cohérents et interprétables, permettant de définir des stratégies d'évacuation multi-scénario théoriquement optimales.

Dans nos simulations de situations d'évacuation non perturbées, les temps d'évacuation obtenus ont confirmé la validité de notre modèle. Nous avons également pu mettre en valeur le cas de la sortie Méano Nord, démontrant son rôle capital dans le processus d'évacuation. La tribune Meano et donc l'ensemble du stade sont évacués bien plus rapidement lorsque celle-ci est ouverte et utilisée. Notre étude souligne par ailleurs l'importance cruciale des liaisons entre les différentes tribunes et blocs ; leur connexion permet une meilleure répartition sur les différents escaliers et les différentes sorties, et donc un temps d'évacuation bien plus court.

Nous comptons donc transmettre ce rapport et nos conclusions à M. Cédric Uha, responsable de la sécurité du Stade de Reims, et nous tenons à sa disposition pour toute autre simulation ou collaboration éventuelle.

## RÉFÉRENCES

- [1] Grégoire Allaire, Alexandre Ern. 2024. cours de deuxième année à l'École polytechnique *Optimisation et contrôle*
- [2] Frédéric Bonnans, Stéphane Gaubert, 2015. cours de troisième année à l'École polytechnique. *Recherche opérationnelle : aspects mathématiques et applications*
- [3] Pierre Fouilhoux. 2020. cours de deuxième année à l'Université Sorbonne Paris Nord, Institut Galilée, *Optimisation Linéaire*
- [4] Victor Klee, Georges J. Minty. 1972. How good is the simplex algorithm ? *Inequalities, III* Academic Press, New York-London
- [5] L.G Chalmet, R.L Francis, P.B Saunders. 1982. *Network Models for Building Evacuation*
- [6] James B. Orlin, Ravindra K. Ahuja, Thomas L. Magnanti. 1993. *Network Flows : Theory, Algorithms and Applications*
- [7] Martin Skutella. 1993. *An Introduction to Network Flows Over Time*

# A

## ANNEXE

---

```

1 import networkx as nx
2
3 def init_graph(Tmax):
4     Gd = nx.DiGraph()
5
6     Noeuds = []
7     Repartition = [] #effectif de d part dans les tribunes
8     Capacites = [] #capacit des noeuds
9     Arcs = []
10
11     ## Noeuds, repartition et capacites
12
13 # Jonquet
14     blocs_bas_Jonquet = ["Bloc 16", "Bloc 17", "Bloc 18", "Bloc 19"]
15     capacites_tribunes_bas_Jonquet = [779, 646, 664, 567]
16     repartition_bas_Jonquet = [779, 646, 664, 567]
17
18     blocs_hauts_Jonquet = ["Bloc 37", "Bloc 38", "Bloc 39", "Bloc 40", "Bloc 41"]
19     repartition_haut_Jonquet = [459, 480, 444, 456, 411]
20     capacites_tribunes_haut_Jonquet = [459, 480, 444, 456, 411]
21
22     coursives_Jonquet=[ "J1", "J2", "J3", "J4", "J5"]
23     repartition_coursives_Jonquet = [0, 0, 0, 0, 0]
24     capacites_coursives_Jonquet = [530, 530, 530, 530, 530]
25
26     sorties_Jonquet = [ "sortie_1_J", "sortie_2_J", "sortie_3_J"]
27     capacites_sorties_Jonquet = [50000, 50000, 50000]
28     repartition_sorties_Jonquet = [0, 0, 0]
29
30     Jonquet = blocs_bas_Jonquet + blocs_hauts_Jonquet + coursives_Jonquet +
31     sorties_Jonquet
32     capacites_Jonquet = repartition_bas_Jonquet + repartition_haut_Jonquet +
33     + capacites_coursives_Jonquet + capacites_sorties_Jonquet
34     repartition_Jonquet = repartition_bas_Jonquet +
35     repartition_haut_Jonquet + repartition_coursives_Jonquet +
36     repartition_sorties_Jonquet
37
38 # Meano
39     blocs_bas_Meano=["Bloc 11", "Bloc 12", "Bloc 13", "Bloc 14", "Bloc 15"]
40     blocs_hauts_Meano=["Bloc 31", "Bloc 32", "Bloc 33", "Bloc 34", "Bloc 35",
41     , "Bloc 36"]
42     capacites_blocs_Meano = [927, 693, 661, 677, 831, 423, 482, 484, 494,
43     471, 422]
44     repartition_blocs_Meano = [927, 693, 661, 677, 831, 423, 482, 484, 494,
45     471, 422]

```

```

39
40 coursives_Meano = ["M0", "M1", "M2", "M3", "M4", "M5", "M6"]
41 capacites_coursives_Meano = [915, 915, 915, 915, 915, 915, 915]
42 repartition_coursives_Meano = [0, 0, 0, 0, 0, 0, 0]
43
44 sorties_Meano = ["sortie_1_m"]
45 capacites_sorties_Meano = [50000]
46 repartition_sorties_Meano = [0]
47
48
49 Meano = blocs_bas_Meano + blocs_hauts_Meano + coursives_Meano +
50 sorties_Meano
51 capacites_Meano = capacites_blocs_Meano + capacites_coursives_Meano +
52 capacites_sorties_Meano
53 repartition_Meano = repartition_blocs_Meano +
54 repartition_coursives_Meano + repartition_sorties_Meano
55
56 # Germain
57 blocs_bas_Germain = ["Bloc "+str(i) for i in range(1,7)]
58 blocs_hauts_Germain = ["Bloc 20", "Bloc 21", "Bloc 22", "Bloc 23", "Bloc 24",
59 , "Bloc 25"]
60 capacites_blocs_Germain = [717, 119, 291, 422, 121, 717, 418, 395, 186,
61 174, 395, 537]
62 repartition_blocs_Germain = [717, 119, 291, 422, 121, 717, 418, 395,
63 186, 174, 395, 537]
64
65 coursives_Germain = ["G1", "G2", "G3", "G4", "G5", "G6", "G7", "G8"]
66 capacites_coursives_Germain = [50000, 50000, 50000, 50000, 50000, 50000,
67 50000, 50000]
68 repartition_coursives_Germain = [0, 0, 0, 0, 0, 0, 0, 0]
69
70 sorties_Germain = ['sortie_1_G']
71 capacites_sorties_Germain = [50000]
72 repartition_sorties_Germain = [0]
73
74 Germain = blocs_bas_Germain + blocs_hauts_Germain + coursives_Germain +
75 sorties_Germain
76 capacites_Germain = capacites_blocs_Germain +
77 capacites_coursives_Germain + capacites_sorties_Germain
78 repartition_Germain = repartition_blocs_Germain +
79 repartition_coursives_Germain + repartition_sorties_Germain
80
81 # Batteux
82 blocs_bas_Batteux = ["Bloc "+str(i) for i in [7,8,9]]
83 blocs_hauts_Batteux = ["Bloc "+str(i) for i in [26,27,28]]
84 capacites_blocs_batteux = [582, 641, 545, 666, 527, 455]
85 repartition_blocs_batteux = [582, 641, 545, 666, 527, 455]
86
87 coursives_Batteux = ["B4", "B5", "B6", "B7"]
88 capacites_coursives_Batteux = [350, 350, 350, 350]
89 repartition_coursives_Batteux = [0, 0, 0, 0]

```

```

1 blocs_visiteurs_bas_Batteux = ["Bloc 10"]
2 blocs_visiteurs_haut_Batteux = ["Bloc 29", "Bloc 30"]
3 capacites_visiteurs_batteux = [419, 232, 106]
4 repartition_visiteurs_batteux = [419, 232, 106]
5
6 coursives_visiteurs_Batteux = ["B1", "B2", "B3"]
7 capacites_coursives_visiteurs_Batteux = [350, 350, 350]
8 repartition_coursives_visiteurs_Batteux = [0, 0, 0]
9
10 sorties_Batteux = ["sortie_"+str(i)+"_b" for i in range(1,3)]
11 capacites_sorties_Batteux = [50000, 50000]
12 repartition_sorties_Batteux = [0, 0]
13
14
15 Batteux = blocs_bas_Batteux + blocs_hauts_Batteux + coursives_Batteux +
16 blocs_visiteurs_bas_Batteux + blocs_visiteurs_haut_Batteux +
17 coursives_visiteurs_Batteux + sorties_Batteux
18 capacites_Batteux = capacites_blocs_batteux +
19 capacites_coursives_Batteux + capacites_visiteurs_batteux +
20 capacites_coursives_visiteurs_Batteux + capacites_sorties_Batteux
21 repartition_Batteux = repartition_blocs_batteux +
22 repartition_coursives_Batteux + repartition_visiteurs_batteux +
23 repartition_coursives_visiteurs_Batteux + repartition_sorties_Batteux
24
25 ## Total
26 Noeuds = Jonquet + Meano + Germain + Batteux
27 Capacites = capacites_Jonquet + capacites_Meano + capacites_Germain +
28 capacites_Batteux
29 Repartition = repartition_Jonquet + repartition_Meano +
30 repartition_Germain + repartition_Batteux
31
32 N = 0
33 for rep in Repartition:
34     N += rep
35
36 Gd.add_node("Sortie", demand = N)
37
38 Capacites.append(N)
39
40
41
42 pos = {"B1": (3, 0), "B2": (4, 0), "B3": (5, 0), 10: (4.5, 0.5), 30:
43 (4, 1), 29: (5, 1), "B4": (7, 0), "B5": (8, 0), "B6": (9, 0), "B7": (10, 0),
44 "Bloc 9": (7, 0.5), "Bloc 8": (8, 0.5), "Bloc 7": (9, 0.5), "Bloc 28":
45 (7, 1), "Bloc 27": (8, 1), "Bloc 26": (9, 1), "sortie_1_b": (6, -1),
46 "Bloc 10": (4.5, 0.5), "Bloc 30": (4, 1), "Bloc 29": (5, 1), "sortie_1_m":
47 (1, 1),
48 "M6": (0, 9), "M5": (0, 8), "M4": (0, 7), "M3": (0, 6), "M2": (0, 5), "M1":
49 (0, 4), "M0": (0, 3),
50 "Bloc 15": (0.5, 7.5), "Bloc 14": (0.5, 6.5), "Bloc 13": (0.5, 5.5), "Bloc
51 12": (0.5, 4.5), "Bloc 11": (0.5, 3.5),
52 "Bloc 1": (0.5, 2.5), "Bloc 2": (0.5, 1.5), "Bloc 3": (0.5, 0.5), "Sortie": (0, 0)
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
```

```

120     "Bloc 36": (1,8), "Bloc 35": (1,7), "Bloc 34": (1,6), "Bloc 33":  

121     (1,5), "Bloc 32": (1,4), "Bloc 31": (1,3), "J5": (0.5,10.5),  

122     "Bloc 37": (4,10), "Bloc 38": (5,10), "Bloc 39": (6,10), "Bloc 40":  

123     (7,10), "Bloc 41": (8,10),  

124     "Bloc 16": (4.5,10.5), "Bloc 17": (5.5,10.5), "Bloc 18":  

125     (6.5,10.5), "Bloc 19": (7.5,10.5), "J4": (4,11), "J3": (5,11), "J2":  

126     (6,11), "J1": (7,11),  

127     "sortie_1_J": (4.5,12), "sortie_2_J": (5.5,12), "sortie_3_J":  

128     (1,11),  

129     "Bloc 25": (10,3), "Bloc 24": (10,4), "Bloc 23": (10,5), "Bloc  

130     22": (10,6), "Bloc 21": (10,7), "Bloc 20": (10,8), "G6": (10.5,4.5), "G7":  

131     (10.5,6.5),  

132     "Bloc 1": (11,8), "Bloc 2": (11,7), "Bloc 3": (11,6), "Bloc 4":  

133     (11,5), "Bloc 5": (11,4), "Bloc 6": (11,3),  

134     "G5": (11.5,8.5), "G4": (11.5,7.5), "G3": (11.5,6.5), "G8":  

135     (12,5.5), "G2": (11.5,4.5), "G1": (11.5,3.5), "sortie_2_b": (11.5,1), "  

136     sortie_1_G": (11,11), "Sortie": (11,12)}  

137  

138 ## Arcs  

139  

140 Arcs = [  

141     ['Bloc 16', 'J4', 9, 3, 0],  

142     ['Bloc 16', 'J5', 9, 3, 0],  

143     ['Bloc 17', 'J3', 9, 3, 0],  

144     ['Bloc 18', 'J2', 9, 3, 0],  

145     ['Bloc 19', 'J1', 9, 3, 0],  

146     ['Bloc 37', 'J4', 6, 6, 0],  

147     ['Bloc 38', 'J3', 9, 6, 0],  

148     ['Bloc 39', 'J2', 6, 6, 0],  

149     ['Bloc 40', 'J1', 6, 6, 0],  

150     ['Bloc 37', 'Bloc 38', 16, 6, 0],  

151     ['Bloc 38', 'Bloc 37', 16, 6, 0],  

152     ['Bloc 38', 'Bloc 39', 16, 6, 0],  

153     ['Bloc 39', 'Bloc 38', 16, 6, 0],  

154     ['Bloc 39', 'Bloc 40', 16, 6, 0],  

155     ['Bloc 40', 'Bloc 39', 16, 6, 0],  

156     ['Bloc 40', 'Bloc 41', 16, 6, 0],  

157     ['Bloc 41', 'Bloc 40', 16, 6, 0],  

158     ['Bloc 16', 'Bloc 17', 14, 6, 0],  

159     ['Bloc 17', 'Bloc 16', 13, 6, 0],  

160     ['Bloc 17', 'Bloc 18', 13, 6, 0],  

161     ['Bloc 18', 'Bloc 17', 13, 6, 0],  

162     ['Bloc 18', 'Bloc 19', 13, 6, 0],  

163     ['Bloc 19', 'Bloc 18', 13, 6, 0],  

164     ['J1', 'J2', 62, 5, 0],  

165     ['J2', 'J1', 62, 5, 0],  

166     ['J2', 'J3', 62, 5, 0],  

167     ['J3', 'J2', 62, 5, 0],  

168     ['J4', 'J3', 62, 5, 0],  

169     ['J3', 'J4', 62, 5, 0],

```

```

162 # ['J4', 'J5', 50000, 5, 0],
163 # ['J5', 'J4', 50000, 5, 0],
164 ['J5', 'sortie_3_J', 28, 3, 0],
165 ['J4', 'sortie_1_J', 15, 3, 0],
166 ['J3', 'sortie_2_J', 15, 7, 0],
167 ['sortie_1_J', 'Sortie', 46, 0, 1],
168 ['sortie_2_J', 'Sortie', 46, 0, 1],
169 ['sortie_3_J', 'Sortie', 58, 0, 0],
170 ['Bloc 12', 'Bloc 13', 24, 6, 0],
171 ['Bloc 13', 'Bloc 12', 24, 6, 0],
172 ['Bloc 13', 'Bloc 14', 24, 6, 0],
173 ['Bloc 14', 'Bloc 13', 24, 6, 0],
174 ['Bloc 31', 'Bloc 32', 19, 6, 0],
175 ['Bloc 32', 'Bloc 31', 19, 6, 0],
176 ['Bloc 32', 'Bloc 33', 19, 6, 0],
177 ['Bloc 33', 'Bloc 32', 19, 6, 0],
178 ['Bloc 33', 'Bloc 34', 19, 6, 0],
179 ['Bloc 34', 'Bloc 33', 19, 6, 0],
180 ['Bloc 34', 'Bloc 35', 19, 6, 0],
181 ['Bloc 35', 'Bloc 34', 19, 6, 0],
182 ['Bloc 35', 'Bloc 36', 19, 6, 0],
183 ['Bloc 36', 'Bloc 35', 19, 6, 0],
184 ['Bloc 11', 'M1', 9, 3, 0],
185 ['Bloc 11', 'M2', 9, 5, 0],
186 ['Bloc 31', 'M0', 6, 12, 0],
187 ['Bloc 12', 'M3', 9, 5, 0],
188 ['Bloc 13', 'M4', 9, 5, 0],
189 ['Bloc 14', 'M5', 9, 5, 0],
190 ['Bloc 32', 'M1', 9, 12, 0],
191 ['Bloc 33', 'M2', 9, 12, 0],
192 ['Bloc 34', 'M3', 9, 12, 0],
193 ['Bloc 35', 'M4', 6, 12, 0],
194 ['Bloc 36', 'M5', 6, 12, 0],
195 ['Bloc 15', 'M6', 9, 5, 0],
196 ['M0', 'M1', 37, 5, 0],
197 ['M1', 'M0', 37, 5, 0],
198 ['M1', 'M2', 37, 5, 0],
199 ['M2', 'M1', 37, 5, 0],
200 ['M2', 'M3', 37, 5, 0],
201 ['M3', 'M2', 37, 5, 0],
202 ['M3', 'M4', 37, 5, 0],
203 ['M4', 'M3', 37, 5, 0],
204 ['M4', 'M5', 37, 5, 0],
205 ['M5', 'M4', 37, 5, 0],
206 ['M5', 'M6', 37, 5, 0],
207 ['M6', 'M5', 37, 5, 0],
208 ['Bloc 15', 'J5', 9, 3, 0],
209 ['M6', 'J5', 37, 5, 0],
210 ['J5', 'M6', 62, 5, 0],
211 ## ['J5', 'J6', 62, 1, 0],
212 ## ['J6', 'J5', 62, 1, 0],
213 ['Bloc 1', 'Bloc 2', 25, 6, 0],

```

```

214 ['Bloc 2', 'Bloc 1', 25, 6, 0],
215 ['Bloc 5', 'Bloc 6', 25, 6, 0],
216 ['Bloc 6', 'Bloc 5', 25, 6, 0],
217 ['Bloc 20', 'Bloc 21', 15, 6, 0],
218 ['Bloc 21', 'Bloc 20', 15, 6, 0],
219 ['Bloc 20', 'G5', 9, 6, 0],
220 ['Bloc 21', 'Bloc 22', 15, 6, 0],
221 ['Bloc 22', 'Bloc 21', 15, 6, 0],
222 ['Bloc 23', 'Bloc 24', 15, 6, 0],
223 ['Bloc 24', 'Bloc 23', 15, 6, 0],
224 ['Bloc 24', 'Bloc 25', 15, 6, 0],
225 ['Bloc 25', 'Bloc 24', 15, 6, 0],
226 ['Bloc 25', 'B7', 9, 6, 0],
227 ['Bloc 6', 'G1', 10, 3, 0],
228 #['Bloc 6', 'B7', 34, 6, 0],
229 ['G1', 'G2', 34, 4, 0],
230 ['G2', 'G1', 34, 4, 0],
231 ['Bloc 5', 'G2', 10, 4, 0],
232 ['Bloc 2', 'G3', 10, 4, 0],
233 ['G3', 'G4', 34, 4, 0],
234 ['G4', 'G3', 34, 4, 0],
235 ['G4', 'G5', 34, 5, 0],
236 ['G5', 'G4', 34, 5, 0],
237 ['Bloc 1', 'G4', 10, 3, 0],
238 ['Bloc 1', 'G5', 10, 3, 0],
239 ['G5', 'J1', 42, 5, 0],
240 ['J1', 'G5', 42, 5, 0],
241 ['Bloc 21', 'G7', 9, 1, 0],
242 ['Bloc 22', 'G7', 9, 1, 0],
243 ['G7', 'G3', 8, 8, 0],
244 ['Bloc 41', 'G5', 11, 4, 0],
245 ['G5', 'sortie_1_G', 25, 3, 0],
246 ['sortie_1_G', 'Sortie', 50000, 0, 1],
247 ['G2', 'G8', 34, 4, 0],
248 ['G8', 'G2', 34, 4, 0],
249 ['G8', 'G3', 34, 4, 0],
250 ['G3', 'G8', 34, 4, 0],
251 ['Bloc 3', 'G8', 10, 8, 0],
252 ['Bloc 4', 'G8', 10, 8, 0],
253 ['Bloc 2', 'Bloc 3', 25, 6, 0],
254 ['Bloc 3', 'Bloc 2', 25, 6, 0],
255 ['Bloc 3', 'Bloc 4', 25, 6, 0],
256 ['Bloc 4', 'Bloc 3', 25, 6, 0],
257 ['Bloc 4', 'Bloc 5', 25, 6, 0],
258 ['Bloc 5', 'Bloc 4', 25, 6, 0],
259 ['G6', 'G2', 8, 8, 0],
260 ['Bloc 23', 'G6', 9, 1, 0],
261 ['Bloc 24', 'G6', 9, 1, 0],
262 ['Bloc 7', 'Bloc 8', 23, 6, 0],
263 ['Bloc 8', 'Bloc 7', 23, 6, 0],
264 ['Bloc 8', 'Bloc 9', 23, 6, 0],
265 ['Bloc 9', 'Bloc 8', 23, 6, 0],

```

```

266 ['Bloc 26', 'Bloc 27', 16, 6, 0],
267 ['Bloc 27', 'Bloc 26', 16, 6, 0],
268 ['Bloc 27', 'Bloc 28', 16, 6, 0],
269 ['Bloc 28', 'Bloc 27', 16, 6, 0],
270 ['B1', 'B2', 25, 3, 0],
271 ['B2', 'B1', 25, 3, 0],
272 ['Bloc 10', 'B2', 8, 4, 0],
273 ##['Bloc 10', 'B1', 9, 1, 0],
274 ['B2', 'B3', 25, 2, 0],
275 ['B3', 'B2', 25, 2, 0],
276 ['Bloc 30', 'B2', 6, 6, 0],
277 ['Bloc 29', 'B3', 6, 6, 0],
278 ['Bloc 29', 'Bloc 30', 11, 6, 0],
279 ['Bloc 30', 'Bloc 29', 11, 6, 0],
280 ['Bloc 9', 'B4', 10, 4, 0],
281 ['B4', 'B5', 32, 5, 0],
282 ['B5', 'B4', 32, 5, 0],
283 ['Bloc 28', 'B5', 9, 6, 0],
284 ['Bloc 8', 'B5', 10, 4, 0],
285 ['B5', 'B6', 32, 5, 0],
286 ['B6', 'B5', 32, 5, 0],
287 ['Bloc 27', 'B6', 9, 6, 0],
288 ['Bloc 7', 'B6', 10, 4, 0],
289 ['B6', 'B7', 32, 5, 0],
290 ['B7', 'B6', 32, 5, 0],
291 ['Bloc 26', 'B7', 9, 6, 0],
292 ['B7', 'G1', 37, 5, 0],
293 ['G1', 'B7', 37, 5, 0],
294 ['B7', 'sortie_2_b', 15, 5, 0],
295 ['B2', 'sortie_1_b', 15, 4, 0],
296 ['sortie_1_b', 'Sortie', 50000, 0, 1],
297 ['sortie_2_b', 'Sortie', 50000, 0, 1],
298 ['sortie_1_m', 'Sortie', 50000, 0, 1],
299 ['M0', 'sortie_1_m', 37, 5, 0]
300 ]
301
302 ## for arc in list(Arcs): # Utiliser list(Arcs) pour éviter de modifier la
303 # liste pendant l'itération
304 ## if (arc[1]!="Sortie"):
305 ##     Arcs.append([arc[1], arc[0], arc[2], arc[3], arc[4]])
306
307
308
309 for t in range(Tmax):
310     for noeud, capacite, rep in zip(Noeuds, Capacites, Repartition):
311         if (t==0):
312             Gd.add_node(noeud+"-"+str(t)+"-1", demand = -rep)
313             Gd.add_node(noeud+"-"+str(t)+"-2")
314             Gd.add_edge(noeud+"-"+str(t)+"-1", noeud+"-"+str(t)+"-2",
315             capacity = capacite, weight = 0)
316         else:

```

```

316     Gd.add_node(noeud+"-"+str(t)+"-1")
317     Gd.add_node(noeud+"-"+str(t)+"-2")
318     Gd.add_edge(noeud+"-"+str(t)+"-1", noeud+"-"+str(t)+"-2",
319         capacity = capacite, weight = 0)
320         if (t>0):
321             Gd.add_edge(noeud+"-"+str(t-1)+"-2", noeud+"-"+str(t)+"-1",
322         capacity = capacite, weight = 0)
323         for arc in Arcs:
324             if (t>=arc[3]):
325                 if (arc[1] == 'Sortie'):
326                     Gd.add_edge(arc[0]+"-"+str(t-arc[3])+"-2", 'Sortie',
327         capacity = arc[2], weight = (N+1)**t)
328             else:
329                 Gd.add_edge(arc[0]+"-"+str(t-arc[3])+"-2", arc[1]+"-"++
330             str(t)+"-1", capacity = arc[2], weight = 1)
331
332     return Noeuds, Capacites, Repartition, pos, Arcs, Gd

```

```

1 import networkx as nx
2 import matplotlib.pyplot as plt
3 import pandas as pd
4 import numpy as np
5 import math
6
7 import Graphe
8
9 import pygame
10
11 from pygame.locals import *
12
13 import cv2
14 from cv2 import COLOR_RGBA2BGR
15 import sys
16
17 Tmax = 300
18 Noeuds, Capacites, Repartition, pos, Arcs, Gd = Graphe.init_graph(Tmax)
19
20
21 ## Calcul par l'algorithme de coût minimum
22
23 flow_cost, flow_dict = nx.network_simplex(Gd)
24
25 print(f"Coût total: {flow_cost}")
26 print("Flot par arc:")
27 for origine in flow_dict:
28     for destination, flot in flow_dict[origine].items():
29         if flot > 0:
30             print(f"{origine} -> {destination}: {flot}")
31
32 colonnes_temps = range(Tmax)
33
34 arcs = [arc[0] + " --> " + arc[1] for arc in Arcs]

```

```

35 df = pd.DataFrame(0, index=Noeuds + arcs, columns=colonnes_temps)
36
37 for origine in flow_dict:
38     ori = origine.split('-')
39     for destination, flot in flow_dict[origine].items():
40         des = destination.split('-')
41         if (ori[0] == des[0]) and (ori[1] == des[1]):
42             df.loc[ori[0], int(ori[1])] += flot
43         elif (ori[0] != des[0] and des[0] != "Sortie"):
44             for t in range(int(des[1])-int(ori[1])):
45                 df.loc[ori[0] + " --> " + des[0], int(ori[1])+t+1] += flot
46
47 print(df)
48
49
50 import os
51 chemin_fichier_excel = 'PSC.xlsx'
52 df.to_excel(chemin_fichier_excel, index=True)
53
54 df = pd.read_excel("PSC.xlsx", index_col=0)
55
56 # Creez un gradient de couleurs
57 fig, ax = plt.subplots(figsize=(20, 5))
58 fig.subplots_adjust(bottom=0.5)
59
60 cmap = plt.get_cmap('RdYlGn_r')
61 norm = plt.Normalize(vmin=0, vmax=1)
62
63 cb1 = plt.colorbar(plt.cm.ScalarMappable(norm=norm, cmap=cmap), cax=ax,
64     orientation='horizontal')
65 cb1.set_label('Taux de remplissage des noeuds et des arcs : Rouge designant
une saturation', fontsize = 20)
66 plt.savefig('color_scale.png')
67 plt.close()
68 color_scale = pygame.image.load('color_scale.png') # Chargement de l'image
69
70 def run():
71     FPS = 40 #Vitesse d'animation
72     RECORDING = True
73     VIDEO = []
74     Pos_pygame_noeuds={Noeuds[i]: [3*50*(pos[Noeuds[i]][0] + 0.75), 3*80*(pos
75 [Noeuds[i]][1]+2)] for i in range(len(Noeuds))}
76     pygame.init()
77     fpsClock=pygame.time.Clock()
78
79     # Taille de l'écran d'affichage
80     SCREEN_WIDTH, SCREEN_HEIGHT = 3*1200, 3*650
81     SCREEN_WIDTH_2, SCREEN_HEIGHT_2 = 3*1200, 3*650
82     screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT), 0, 32)
83
84     rayon_noeud=3*14

```

```

84     surface = pygame.Surface(screen.get_size())
85     surface = surface.convert()
86
87     class Node(object):
88         #Classe des objets noeuds(=cercle)
89         def __init__(self, id, pos_x, pos_y, capacity):
90             self.font = pygame.font.Font(None, 4*10) #Taille de la legende
91             du texte
92             self.id=id # Nom du noeud
93             self.radius= rayon_noeud #Rayon du cercle du noeud
94             self.capacity=capacity
95             self.pos=(pos_y, pos_x) #Position du noeud
96
97             def draw(self, surf, t):
98                 taux=df[t][self.id]/self.capacity
99                 color = [c*255 for c in cmap(taux)[0:3]]
100                pygame.draw.circle(surf, color, self.pos,self.radius, width=0) #Trace le cercle, width=0 indique qu'on rempli le cercle avec la couleur
101                pygame.draw.circle(surf, color, self.pos,self.radius, width=1) #Trace le contour
102                surface.blit(self.font.render(str(self.id), True, (0,0,0)), (
103                  self.pos[0]-self.radius/4, self.pos[1]-self.radius/5, 40, 40)) #Ecris le
104                  nom du noeud au centre du cercle
105
106      class Edge(object):
107          #Classe des objets arcs(=traits)
108          def __init__(self, start, end, capacity):
109              self.start=start # Nom du noeud
110              self.end=end
111              self.capacity=capacity
112
113              def draw(self, surf, t):
114                  taux=df[t][self.start+ ' --> ' + self.end]/self.capacity
115                  color = [c*255 for c in cmap(taux)[0:3]]
116                  x_A =(Pos_pygame_noeuds[self.start][1])
117                  x_B =Pos_pygame_noeuds[self.end][1]
118                  y_A = Pos_pygame_noeuds[self.start][0]
119                  y_B = Pos_pygame_noeuds[self.end][0]
120                  d = math.sqrt((x_B-x_A)**2 + (y_B-y_A)**2)
121                  pygame.draw.line(surf, color, (Pos_pygame_noeuds[self.start][1],
122                  Pos_pygame_noeuds[self.start][0]), (Pos_pygame_noeuds[self.end][1],
123                  Pos_pygame_noeuds[self.end][0]), 3)
124
125      if __name__ == '__main__':
126          time_limit=Tmax #Horizon de discréétisation du temps
127          Nodes=[Node(Noeuds[i], Pos_pygame_noeuds[Noeuds[i]][0],
128          Pos_pygame_noeuds[Noeuds[i]][1], Capacites[i]) for i in range(len(Noeuds))
129          ] #Liste des noeuds
130
131
132          Edges=[Edge(Arcs[i][0], Arcs[i][1], Arcs[i][2]) for i in range(len(
133          Arcs)) if Arcs[i][1]!='Sortie'] #Liste des noeuds

```

```

126
127     time=0
128     running=True
129     stop=False
130     while running and time <Tmax-1:
131         for event in pygame.event.get(): #Ensemble des événements
132             possibles avec les touches du clavier
133             if event.type == pygame.QUIT: #Ferme la fenêtre
134                 running = False
135             elif event.type == KEYDOWN:
136                 if event.key == K_SPACE: #Barre d'espace = faire une
137                     pause ou redemarrer
138                     stop= not stop
139             elif event.key == K_LEFT: # Ralentir l'animation
140                 if FPS >=2:
141                     FPS-=1
142             elif event.key == K_RIGHT: #Accelerer l'animation
143                 FPS+=1
144             elif stop and event.key == K_UP: #Saut dans le futur
145                 time=min(time_limit, time+20)
146             elif stop and event.key == K_DOWN: #Saut dans le passé
147                 time=max(0, time-20)

148             surface.fill((255,255,255)) #Couleur du fond de l'écran (blanc)
149             for noeud in Nodes:
150                 noeud.draw(surface,time) #Trace les noeuds

151             for arc in Edges:
152                 arc.draw(surface,time) #Trace les arcs

153
154             screen.blit(surface, (0,0))
155             font = pygame.font.Font(None, 36)
156             temps_reel = 3*time
157             text = font.render('Temps : ' + str(temps_reel) + ' secondes',
158             True, (20, 20, 20)) #Ecris le temps au cours de l'animation
159             text_rect = text.get_rect(center=(SCREEN_WIDTH // 2,
160             SCREEN_HEIGHT // 2 - 200))
161             surface.blit(text, text_rect)
162             scale_rect = color_scale.get_rect(center=(SCREEN_WIDTH // 2,
163             SCREEN_HEIGHT // 2 + 200))
164             surface.blit(color_scale, scale_rect)
165             screen.blit(surface, (0,0))
166             resized_screen = pygame.transform.scale(screen, (SCREEN_WIDTH_2,
167             SCREEN_HEIGHT_2))
168             pygame.display.flip()
169             pygame.display.update()
170             if not stop:
171                 time+=1
172                 fpsClock.tick(FPS)
173             try :
174                 if RECORDING :
175                     VIDEO.append(pygame.image.tostring(resized_screen,'RGB',

```

```
        False))  
    except MemoryError:  
        print("Error")  
        STOP_GAME = True  
        RECORDING = False  
    if time ==time_limit:  
        while running:  
            for event in pygame.event.get():  
                if event.type == pygame.QUIT:  
                    running = False  
    print('\n [+] VIDEO Object size : ', sys.getsizeof(VIDEO))  
    print('\n [+] RECORDING AVI file ')  
    #Record the video  
    if RECORDING:  
        video = cv2.VideoWriter('All_steps.avi',  
                                cv2.VideoWriter_fourcc('M', 'J', 'P', 'G'),  
                                3,  
                                (SCREEN_WIDTH_2, SCREEN_HEIGHT_2), True)  
        for event in pygame.event.get():  
            pygame.event.clear()  
        for image in VIDEO:  
            image = np.frombuffer(image, np.uint8).reshape(  
                SCREEN_HEIGHT_2, SCREEN_WIDTH_2, 3)  
            image = cv2.cvtColor(image, COLOR_RGBA2BGR)  
            video.write(image)  
  
        cv2.destroyAllWindows()  
        video.release()  
  
    pygame.display.quit()  
    pygame.quit()  
  
run()
```