

# Projet : Gestion des horaires de trains

Ultimatum le dimanche 20 mai à `nicolas.mazzocchi@ulb.ac.be`  
Tout retard sera sanctionné

## 1 Informations pratiques

Ce projet entre dans le cadre du cours INFO-F-302 de l'Université libre de Bruxelles. Son objectif principal est la modélisation et la résolution d'un problème par un SAT solver. Pour ce faire, les étudiants pourront travailler seul ou en binôme. A terme, votre code source et un rapport devront être compressés au format ZIP et envoyé par mail<sup>1</sup> à l'adresse mentionnée ci-dessus.

## 2 Énoncé

Jeunes informaticiens, la société de chemin de fer BCNS à besoin de vos services dans le cadre de l'optimisation horaire pour son future réseau ferroviaire ! Actuellement, la localité de Nadzieja est clairsemée de petites villes mal déservies aux profits de rares métropoles. C'est pourquoi la BCNS lance sa campagne **unitravel** afin de garantir aux usagés un train direct fréquent quelque soit l'origine et la destination du trajet. Plus exactement un trajet **unitravel** assurera que :

1. Pour toute paire de gare  $(g, g')$ , pour chaque fenêtre horaire de durée *TimeWindow*, il existe un train qui dessert  $g$  dans cette fenêtre, puis, plus tard, ce même train desservira  $g'$ , après une durée de trajet d'au plus *TravelDuration*.

Les données *TimeWindow* et *TravelDuration* seront des paramètres donnés dans l'entrée du problème. On distingue deux types de gares : *big* et *small*, ainsi que deux types de train, *fast* et *slow* (omnibus).

Les infrastructures opérationnelles ne sont pas encore connues, toutefois, les normes de sécurité restent inchangées. À savoir :

2. Pour prévenir tout risque de collision, une voie de chemin de fer entre deux gares, appelé segment, peut accueillir au plus un train.
3. Les trains de type *slow* sont des omnibus : ils sont tenus de s'arrêter dans toutes les gares. Les trains de type *fast* ne sont tenus de s'arrêter que dans les gares de type *big*.
4. Afin de permettre aux usagers de monter à bord, tout train desservant une gare doit le faire durant une durée d'au moins *TimeWait*.
5. Les trains doivent respecter les temps de trajet prévus par l'infrastructure entre deux gares. On supposera constant, les temps de trajet entre deux gares, ils seront donc donnés en entrée du problème.

---

1. N'oubliez pas de préciser le.s nom.s du ou des propriétaire.s

6. Une gare ne peut accueillir plus de trains que ce que sa capacité le permet.

Votre challenge, noté TRAIN pour la suite, consiste à écrire une formule Booléenne en utilisant la librairie MiniSAT de C++, telle que le problème d'horaire a une solution si et seulement si la formule est satisfaisable. Cette formule devra donc modéliser correctement le trafic ferroviaire et satisfaire les propriétés citées.

Formellement, une instance du problème TRAIN est donnée par les paramètres suivants.

- *Map* : un graphe dirigé pour lequel on interprétera chaque nœud comme une gare et chaque arc comme une voie de chemin de fer (segment). Les nœuds auront chacun une capacité indiquant combien de trains la gare peut supporter et un type *big* ou *small*. Les arcs auront chacun, une durée indiquant combien de temps un train doit mettre pour le traverser.
- *Slow & Fast* : un nombre de trains de type *slow* et un nombre de trains de type *fast*.
- *TimeSlot* : un nombre de minutes correspondant à la durée totale du planning que vous devrez concevoir. Par exemple, si on fixe  $TimeSlot = 180$ , cela signifie qu'on essaie de concevoir un horaire de train pour trois heures consécutives.
- *TimeWait* : un nombre de minutes correspondant à l'attente minimale d'un train dans une gare qu'il dessert.
- *TimeWindow* : un nombre de minutes correspondant à la fenêtre horaire des trajets **unitravel**.
- *TravelDuration* : un nombre de minutes correspondant à la durée maximale d'un trajet **unitravel**.

La sortie de votre programme sera une fonction qui à chaque minute  $0 \leq m < TimeSlot$  et chaque train  $0 \leq t < Slow + Fast$  retourne une position dans le graphe, c'est-à-dire soit un noeud (une gare) soit un arc (un segment).

### 3 Exemple

Basé sur les fichiers `maps/cycle.txt` et `maps/slow-fast.txt`, nous présentons deux exemples avec leurs solutions, que vous devriez pouvoir reproduire. L'affichage du programme corrigé indique par le symbole **x** lorsque le train correspondant est en gare. Les trajets **unitravel** sont ensuite listés précisant l'heure de départ, l'heure d'arrivée et le train qui réalise le voyage.

**Cycle** Considérons une map formant un cycle de quatre gares à laquelle on alloue un unique train de type *slow*. L'idée est de contraindre le train à tourner en boucle le plus vite possible. Ci-dessous, les paramètres et affichage du programme.

	Train number 0 (slow)	Travels
TIMEWAIT 3		
SLOW 1	A  x.....xxxxxxxxxxx	0:A -- t0 -- 3:B
FAST 0	B  ...xxx.....xxx.....	0:A -- t0 -- 8:C
TRAIN (SLOW+FAST)	C  .....xxx.....	3:B -- t0 -- 6:C
TIMESLOT 26		13:B -- t0 -- 25:A
STATION 3		6:C -- t0 -- 13:B
TRAVELDURATION 12		8:C -- t0 -- 16:A
TIMEWINDOW 14		

**Slow-Fast** Considérons une map pour laquelle une gare *small* connecte deux gares *big*. On alloue à ce réseau, 2 trains de type *slow* et un seul de type *fast*. L'idée est de contraindre le train *fast* à ne pas s'arrêter dans la gare centrale *small*. Ci-dessous, les paramètres et affichage du programme.

TIMEWAIT 3	Train number 0 (slow)	Travels
SLOW 2	A  .....xxx...	0:A -- t2 -- 3:C
FAST 1	B  xxxxxx.....xxxxxx....xx	1:A -- t1 -- 3:B
TRAIN (SLOW+FAST)	C  .....xxx.....	10:A -- t2 -- 13:C
TIMESLOT 26		11:A -- t1 -- 14:B
STATION 3	Train number 1 (slow)	20:A -- t2 -- 23:C
TRAVELDURATION 3	A  xx.....xxx.....	21:A -- t0 -- 24:B
TIMEWINDOW 10	B  ...xxxxxx.....xxx.....xxx.	5:B -- t0 -- 7:C
	C  .....xxx.....	8:B -- t1 -- 10:A
		15:B -- t1 -- 18:C
	Train number 2 (fast)	18:B -- t0 -- 20:A
	A  x.....xxx.....xxx.....	5:C -- t2 -- 8:A
	B  .....	9:C -- t0 -- 12:B
	C  ...xxx.....xxx.....xxx	15:C -- t2 -- 18:A
		19:C -- t1 -- 22:B

## 4 Évaluation

**Code (10pts)** L'implémentation devra respecter l'énoncé dans la transcription des contraintes explicites mais aussi implicites. Par contrainte implicites, on entend les contraintes qui ne sont pas directement données par le problème, mais qui sont liées à votre modélisation (par exemple, la contrainte "au plus une valeur par case" au Sudoku).

Seul le fichier `engine/main.cpp` du squelette qui vous a été transmis devra être modifié. Le code devra être (brièvement) commenté afin que le correcteur puisse retrouver quelles sont les contraintes codées.

**Rapport (10pts)** Le rapport devra contenir d'une part :

- une explication de votre modélisation : quelles variables Booléennes vous avez choisies et le sens que vous leur donnez.
- les formules qui correspondent aux différentes contraintes que vous codez
- éventuellement, si vous avez retiré certaines contraintes car elles sont impliquées par d'autres, le dire et les donner.

D'autre part, le rapport devra comprendre les réponses aux éléments suivants (dont l'implémentation est optionnelle) :

1. si on remplace la contrainte 1 par la contrainte suivante :

Pour toute paire de gare  $(g, g')$ , pour chaque fenêtre horaire de durée *TimeWindow*, il est possible de prendre un train dans la fenêtre horaire, en gare  $g$ , pour aller vers  $g'$  après une durée maximale de trajet *TravelDuration*, le tout en changeant potentiellement de train au plus *NbChange* fois. La version précédente du problème correspond au cas *NbChange*=0.

2. ce problème est une version très simplifiée de la réalité. Proposez deux améliorations rendant le modèle plus réaliste et expliquez comment vous feriez pour les modéliser en SAT.

**Bonus (3pts)** Ajoutez au rapport la démonstration que le problème TRAIN est NP-DUR. Autrement dit, on vous a demandé précédemment de réduire le problème TRAIN au problème SAT. Dans cette question, on vous demande l'inverse.

## 5 Indications

Voici quelques informations concernant le code source qui vous a été transmis.

**Interroger la map** La liste des méthodes de la map est la suivante :

- `int get_duration (int from, int to);`  
retourne le temps entre les gares ayant les identifiants entier respectif *from* et *to*
- `int get_capacity (int node);`  
retourne la capacité de la gare ayant l'identifiant entier *node*
- `Type get_type (int node);`  
retourne le type de la gare ayant l'identifiant entier *node*
- `char *get_name (int node);`  
retourne le nom de la gare ayant l'identifiant entier *node*
- `int get_names_size ();`  
retourne la taille du plus grand nom de gare de la map
- `int get_size ();`  
retourne le nombre de gare de la map
- `void print ();`  
afficher la map

**Utilisez vos propres graphes** Le code source qui vous a été remis dispose d'un parser de fichier pour l'encodage du réseau ferroviaire. Son format est simple.

- une ligne = une gare
- vous pouvez définir une voix de chemin de fer entre une gare,  $g : d$  avec  $g$  la ligne de la gare  $g$  (la numérotation commence à 1) et  $d$  est la durée de l'arc. Notez qu'une voie de chemin de fer d'une gare vers elle même est automatiquement créé avec une durée de 0, non modifiable.
- vous pouvez définir le type d'une gare, B pour *big* (par défaut) et S pour *small*
- vous pouvez définir la capacité d'une gare, #n pour une capacité de  $n$  (1 par défaut)
- vous pouvez définir le nom d'une gare (pour l'affichage), @nom ("?" par défaut).

Attention, pour satisfaire la contrainte (1) votre graphe doit être fortement connecté, c'est à dire que l'on peut aller de n'importe quelle gare vers toutes les autres en empruntant un ou plusieurs arcs. En particulier, si vous ajoutez une ligne vide il y aura une gare qui sera déconnectée des autres.